

### 3.8.2.3-multi-bc-snc-morph-seq Lexical Complexity Binary Classification Prediction Transformers Modeling

April 13, 2025

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

```
[1]: !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
```

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

```
Hit:1 https://cloud.r-project.org/bin/linux/ubuntu jammy-cran40/ InRelease
Hit:2 https://developer.download.nvidia.com/compute/cuda/repos/ubuntu2204/x86_64 InRelease
Hit:3 http://security.ubuntu.com/ubuntu jammy-security InRelease
Hit:4 http://archive.ubuntu.com/ubuntu jammy InRelease
Hit:5 http://archive.ubuntu.com/ubuntu jammy-updates InRelease
Hit:6 https://r2u.stat.illinois.edu/ubuntu jammy 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 32 not upgraded.
```

```
[3]: #@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
```

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

```
[5]: 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).

```
[6]: 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)
```

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

```
[8]: !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

```
[9]: !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
```

```
[10]: !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
```

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

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

loaded_dataframes = {}

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 \*\***

```
[13]: 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

```

```

[14]: 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

```

```

[15]: 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
```

```
[16]: 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

[17]: 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"]
    }
```



```

[18]: 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

```

[19]: # 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 = train_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

```

```

[20]: 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=8
    )

    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 \*\*

```

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

```

```

model checkpoints: /content/drive/MyDrive/266-final/models/
multi_answerdotai
multi_bert-base-cased_binary_complexity_20250408_143322
multi_bert-base-cased_binary_complexity_20250409_175804
multi_bert-base-cased_binary_complexity_20250409_175954
multi_bert-base-cased_binary_complexity_20250409_180139
multi_bert-base-cased_binary_complexity_20250409_185057
multi_bert-base-cased_binary_complexity_20250409_185213
multi_bert-base-cased_binary_complexity_20250409_185333
multi_bert-base-cased_binary_complexity_20250409_234934
multi_bert-base-cased_binary_complexity_20250410_001637
multi_bert-base-cased_binary_complexity_20250410_003117
multi_bert-base-cased_binary_complexity_20250410_004527
multi_bert-base-cased_binary_complexity_20250410_025823
multi_bert-base-cased_binary_complexity_20250410_030623
multi_bert-base-cased_binary_complexity_20250410_031401
multi_bert-base-cased_binary_complexity_20250410_032138
multi_bert-base-cased_binary_complexity_20250410_034203
multi_bert-base-cased_binary_complexity_20250410_034823
multi_bert-base-cased_binary_complexity_20250410_035510
multi_bert-base-cased_binary_complexity_20250410_040140
multi_bert-base-cased_binary_complexity_20250410_174340
multi_bert-base-cased_binary_complexity_20250411_002219

```

multi\_bert-base-cased\_binary\_complexity\_20250411\_044230  
multi\_bert-base-cased\_binary\_complexity\_20250411\_045829  
multi\_bert-base-cased\_binary\_complexity\_20250411\_123321  
multi\_bert-base-cased\_binary\_complexity\_20250411\_123328  
multi\_bert-base-cased\_binary\_complexity\_20250411\_123334  
multi\_bert-base-cased\_binary\_complexity\_2025-04-11T08:45:37.508277-07:00  
multi\_bert-base-cased\_binary\_complexity\_2025-04-11T08:49:23.225636-07:00  
multi\_bert-base-cased\_binary\_complexity\_2025-04-11T08:52:27.244781-07:00  
multi\_bert-base-cased\_binary\_complexity\_2025-04-11T08:58:48.717503-07:00  
multi\_bert-base-cased\_binary\_complexity\_75th\_split\_20250411\_005437  
multi\_bert-large-cased\_binary\_complexity\_20250411\_002650  
multi\_bert-large-cased\_binary\_complexity\_20250411\_044710  
multi\_bert-large-cased\_binary\_complexity\_20250411\_050144  
multi\_bert-large-cased\_binary\_complexity\_20250411\_123609  
multi\_bert-large-cased\_binary\_complexity\_20250411\_123628  
multi\_bert-large-cased\_binary\_complexity\_20250411\_123925  
multi\_bert-large-cased\_binary\_complexity\_2025-04-11T08:48:18.745348-07:00  
multi\_bert-large-cased\_binary\_complexity\_2025-04-11T08:54:05.820676-07:00  
multi\_bert-large-cased\_binary\_complexity\_2025-04-11T09:02:02.177069-07:00  
multi\_bert-large-cased\_binary\_complexity\_75th\_split\_20250411\_010152  
multi\_microsoft  
multi\_roberta-base\_binary\_complexity\_20250411\_002307  
multi\_roberta-base\_binary\_complexity\_20250411\_044250  
multi\_roberta-base\_binary\_complexity\_20250411\_045856  
multi\_roberta-base\_binary\_complexity\_20250411\_123340  
multi\_roberta-base\_binary\_complexity\_20250411\_123350  
multi\_roberta-base\_binary\_complexity\_20250411\_123353  
multi\_roberta-base\_binary\_complexity\_20250411\_154623  
multi\_roberta-base\_binary\_complexity\_20250411\_154942  
multi\_roberta-base\_binary\_complexity\_20250411\_155244  
multi\_roberta-base\_binary\_complexity\_20250411\_155935  
multi\_roberta-base\_binary\_complexity\_75th\_split\_20250411\_005524  
multi\_roberta-large\_binary\_complexity\_20250411\_002759  
multi\_roberta-large\_binary\_complexity\_20250411\_044824  
multi\_roberta-large\_binary\_complexity\_20250411\_050222  
multi\_roberta-large\_binary\_complexity\_20250411\_123652  
multi\_roberta-large\_binary\_complexity\_20250411\_123711  
multi\_roberta-large\_binary\_complexity\_20250411\_124008  
multi\_roberta-large\_binary\_complexity\_20250411\_154930  
multi\_roberta-large\_binary\_complexity\_20250411\_155444  
multi\_roberta-large\_binary\_complexity\_20250411\_160310  
multi\_roberta-large\_binary\_complexity\_75th\_split\_20250411\_010302  
multi\_xlnet  
single\_answerdotai  
single\_bert-base-cased\_binary\_complexity\_20250408\_043117  
single\_bert-base-cased\_binary\_complexity\_20250408\_043334  
single\_bert-base-cased\_binary\_complexity\_20250408\_043750  
single\_bert-base-cased\_binary\_complexity\_20250409\_175702

single\_bert-base-cased\_binary\_complexity\_20250409\_175900  
single\_bert-base-cased\_binary\_complexity\_20250409\_180045  
single\_bert-base-cased\_binary\_complexity\_20250409\_185027  
single\_bert-base-cased\_binary\_complexity\_20250409\_185141  
single\_bert-base-cased\_binary\_complexity\_20250409\_185303  
single\_bert-base-cased\_binary\_complexity\_20250409\_234236  
single\_bert-base-cased\_binary\_complexity\_20250410\_000508  
single\_bert-base-cased\_binary\_complexity\_20250410\_002813  
single\_bert-base-cased\_binary\_complexity\_20250410\_004230  
single\_bert-base-cased\_binary\_complexity\_20250410\_025214  
single\_bert-base-cased\_binary\_complexity\_20250410\_030435  
single\_bert-base-cased\_binary\_complexity\_20250410\_031211  
single\_bert-base-cased\_binary\_complexity\_20250410\_031404  
single\_bert-base-cased\_binary\_complexity\_20250410\_031948  
single\_bert-base-cased\_binary\_complexity\_20250410\_034334  
single\_bert-base-cased\_binary\_complexity\_20250410\_035314  
single\_bert-base-cased\_binary\_complexity\_20250410\_035940  
single\_bert-base-cased\_binary\_complexity\_20250410\_173757  
single\_bert-base-cased\_binary\_complexity\_20250410\_173911  
single\_bert-base-cased\_binary\_complexity\_20250410\_174027  
single\_bert-base-cased\_binary\_complexity\_20250410\_175501  
single\_bert-base-cased\_binary\_complexity\_20250410\_210219  
single\_bert-base-cased\_binary\_complexity\_20250410\_213212  
single\_bert-base-cased\_binary\_complexity\_20250410\_214441  
single\_bert-base-cased\_binary\_complexity\_20250410\_214546  
single\_bert-base-cased\_binary\_complexity\_20250410\_214659  
single\_bert-base-cased\_binary\_complexity\_20250411\_044221  
single\_bert-base-cased\_binary\_complexity\_20250411\_044245  
single\_bert-base-cased\_binary\_complexity\_20250411\_120751  
single\_bert-base-cased\_binary\_complexity\_20250411\_120754  
single\_bert-base-cased\_binary\_complexity\_20250411\_120814  
single\_bert-base-cased\_binary\_complexity\_2025-04-11T08:05:13.902582-07:00  
single\_bert-base-cased\_binary\_complexity\_2025-04-11T08:06:17.534030-07:00  
single\_bert-base-cased\_binary\_complexity\_2025-04-11T08:07:27.352056-07:00  
single\_bert-base-cased\_binary\_complexity\_2025-04-11T08:26:04.979595-07:00  
single\_bert-base-cased\_binary\_complexity\_2025-04-11T08:26:08.572701-07:00  
single\_bert-base-cased\_binary\_complexity\_2025-04-11T08:26:30.856843-07:00  
single\_bert-base-cased\_binary\_complexity\_2025-04-11T08:26:54.451746-07:00  
single\_bert-base-cased\_binary\_complexity\_2025-04-11T08:46:21.496737-07:00  
single\_bert-base-cased\_binary\_complexity\_75th\_split\_20250411\_005451  
single\_bert-large-cased\_binary\_complexity\_20250410\_215725  
single\_bert-large-cased\_binary\_complexity\_20250410\_222431  
single\_bert-large-cased\_binary\_complexity\_20250411\_044617  
single\_bert-large-cased\_binary\_complexity\_20250411\_044715  
single\_bert-large-cased\_binary\_complexity\_20250411\_121110  
single\_bert-large-cased\_binary\_complexity\_20250411\_121219  
single\_bert-large-cased\_binary\_complexity\_20250411\_121349  
single\_bert-large-cased\_binary\_complexity\_2025-04-11T08:09:12.255934-07:00



single\_bert-large-cased\_binary\_complexity\_2025-04-11T08:09:36.911756-07:00  
single\_bert-large-cased\_binary\_complexity\_2025-04-11T08:10:24.066329-07:00  
single\_bert-large-cased\_binary\_complexity\_2025-04-11T08:29:30.583138-07:00  
single\_bert-large-cased\_binary\_complexity\_2025-04-11T08:29:38.951478-07:00  
single\_bert-large-cased\_binary\_complexity\_2025-04-11T08:32:25.963533-07:00  
single\_bert-large-cased\_binary\_complexity\_2025-04-11T08:50:30.982987-07:00  
single\_bert-large-cased\_binary\_complexity\_75th\_split\_20250411\_010303  
single\_microsoft  
single\_roberta-base\_binary\_complexity\_20250410\_212304  
single\_roberta-base\_binary\_complexity\_20250410\_212514  
single\_roberta-base\_binary\_complexity\_20250410\_213732  
single\_roberta-base\_binary\_complexity\_20250410\_214805  
single\_roberta-base\_binary\_complexity\_20250410\_221944  
single\_roberta-base\_binary\_complexity\_20250411\_044307  
single\_roberta-base\_binary\_complexity\_20250411\_044327  
single\_roberta-base\_binary\_complexity\_20250411\_120834  
single\_roberta-base\_binary\_complexity\_20250411\_120837  
single\_roberta-base\_binary\_complexity\_20250411\_120924  
single\_roberta-base\_binary\_complexity\_2025-04-11T08:05:56.911365-07:00  
single\_roberta-base\_binary\_complexity\_2025-04-11T08:07:02.370250-07:00  
single\_roberta-base\_binary\_complexity\_2025-04-11T08:08:10.921118-07:00  
single\_roberta-base\_binary\_complexity\_2025-04-11T08:26:49.931833-07:00  
single\_roberta-base\_binary\_complexity\_2025-04-11T08:26:52.254114-07:00  
single\_roberta-base\_binary\_complexity\_2025-04-11T08:28:07.892457-07:00  
single\_roberta-base\_binary\_complexity\_2025-04-11T08:29:29.440485-07:00  
single\_roberta-base\_binary\_complexity\_2025-04-11T08:47:34.036076-07:00  
single\_roberta-base\_binary\_complexity\_75th\_split\_20250411\_005603  
single\_roberta-large\_binary\_complexity\_20250410\_221054  
single\_roberta-large\_binary\_complexity\_20250410\_222652  
single\_roberta-large\_binary\_complexity\_20250410\_223030  
single\_roberta-large\_binary\_complexity\_20250410\_223320  
single\_roberta-large\_binary\_complexity\_20250410\_223754  
single\_roberta-large\_binary\_complexity\_20250411\_044805  
single\_roberta-large\_binary\_complexity\_20250411\_044938  
single\_roberta-large\_binary\_complexity\_20250411\_121257  
single\_roberta-large\_binary\_complexity\_20250411\_121440  
single\_roberta-large\_binary\_complexity\_20250411\_121538  
single\_roberta-large\_binary\_complexity\_2025-04-11T08:11:00.629981-07:00  
single\_roberta-large\_binary\_complexity\_2025-04-11T08:11:25.589499-07:00  
single\_roberta-large\_binary\_complexity\_2025-04-11T08:12:13.265025-07:00  
single\_roberta-large\_binary\_complexity\_2025-04-11T08:31:21.884000-07:00  
single\_roberta-large\_binary\_complexity\_2025-04-11T08:31:31.124508-07:00  
single\_roberta-large\_binary\_complexity\_2025-04-11T08:34:43.895671-07:00  
single\_roberta-large\_binary\_complexity\_2025-04-11T08:52:46.787583-07:00  
single\_roberta-large\_binary\_complexity\_75th\_split\_20250411\_010518  
single\_xlnet

```
[22]: # 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("=====")
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 \*\***

[23]: *# 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.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 \*\*

[24]: *# 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%|          | 0/7662 [00:00<?, ? examples/s]

```

```

Map:   0%|          | 0/421 [00:00<?, ? examples/s]

```

```

Map:   0%|          | 0/917 [00:00<?, ? examples/s]

```

[illegible]

```
[25]: # 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)
#####
# 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.",

```



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 :

=====

num\_parameters: 108311810

num\_trainable\_parameters at load: 108311810

=====

model lineage: {'type': 'huggingface\_hub', 'path': 'bert-base-cased',  
'timestamp': '2025-04-11 15:58:27'}

=====

```
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: snc_morph_seq

```

```

[26]: # 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-20-0816cb2c5003>: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>
Downloading builder script: 0%|          | 0.00/4.20k [00:00<?, ?B/s]
Downloading builder script: 0%|          | 0.00/7.56k [00:00<?, ?B/s]
Downloading builder script: 0%|          | 0.00/7.38k [00:00<?, ?B/s]
Downloading builder script: 0%|          | 0.00/6.79k [00:00<?, ?B/s]
<IPython.core.display.HTML object>
Validation metrics: {'eval_loss': 0.7170491814613342, 'eval_accuracy':
0.5151515151515151, 'eval_precision': 0.5151515151515151, 'eval_recall': 1.0,
'eval_f1': 0.68, 'eval_runtime': 5.0726, 'eval_samples_per_second': 19.517,
'eval_steps_per_second': 0.197, 'epoch': 1.0}
Test metrics: {'eval_loss': 0.7107948064804077, 'eval_accuracy':
0.5380434782608695, 'eval_precision': 0.5380434782608695, 'eval_recall': 1.0,

```

```
'eval_f1': 0.6996466431095406, 'eval_runtime': 5.2512,  
'eval_samples_per_second': 35.04, 'eval_steps_per_second': 0.381, 'epoch': 1.0}
```

```
[27]: # 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\_2025-04-11T08:58:48.717503-07:00

<IPython.core.display.HTML object>

EXPERIMENT LOGGED TO:  
/content/drive/MyDrive/266-final/results/experiment\_runs.txt



0.2.4 snc roberta-base regularization\_weight\_decay = 0.5 learning\_rate = 5e-6  
size\_batch = 128 length\_max = 128 num\_epochs = 1

```
[28]: # 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("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="roberta-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%| | 0/1517 [00:00<?, ? examples/s]

Map: 0%| | 0/99 [00:00<?, ? examples/s]

```
Map: 0%|          | 0/184 [00:00<?, ? examples/s]
```

Datasets prepared. Sample from train\_data\_hf:

[illegible]

Loading from Hugging Face model: roberta-base

```
tokenizer_config.json: 0%|          | 0.00/25.0 [00:00<?, ?B/s]
```

```
config.json: 0%|          | 0.00/481 [00:00<?, ?B/s]
```

vocab.json: 0%| | 0.00/899k [00:00&lt;?, ?B/s]

```
merges.txt:  0%|          | 0.00/456k [00:00<?, ?B/s]
```

```
tokenizer.json: 0%|          | 0.00/1.36M [00:00<?, ?B/s]
```

```
model.safetensors: 0%|          | 0.00/499M [00:00<?, ?B/s]
```

Some weights of RobertaForSequenceClassification were not initialized from the model checkpoint at roberta-base and are newly initialized:

```
['classifier.dense.bias', 'classifier.dense.weight', 'classifier.out_proj.bias',  
 'classifier.out_proj.weight']
```

You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.

=====

roberta-base :

=====

```
num_parameters: 124647170
```

```
num trainable parameters at load: 124647170
```

=====

```
model lineage: {'type': 'huggingface_hub', 'path': 'roberta-base', 'timestamp':
```

'2025-04-11 15:59:17'}

=====

[29]: `print(model)`

```
RobertaForSequenceClassification(
  (roberta): RobertaModel(
    (embeddings): RobertaEmbeddings(
      (word_embeddings): Embedding(50265, 768, padding_idx=1)
      (position_embeddings): Embedding(514, 768, padding_idx=1)
      (token_type_embeddings): Embedding(1, 768)
      (LayerNorm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (encoder): RobertaEncoder(
      (layer): ModuleList(
        (0-11): 12 x RobertaLayer(
          (attention): RobertaAttention(
            (self): RobertaSdpaSelfAttention(
              (query): Linear(in_features=768, out_features=768, bias=True)
              (key): Linear(in_features=768, out_features=768, bias=True)
              (value): Linear(in_features=768, out_features=768, bias=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
            (output): RobertaSelfOutput(
              (dense): Linear(in_features=768, out_features=768, bias=True)
              (LayerNorm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
          )
          (intermediate): RobertaIntermediate(
            (dense): Linear(in_features=768, out_features=3072, bias=True)
            (intermediate_act_fn): GELUActivation()
          )
          (output): RobertaOutput(
            (dense): Linear(in_features=3072, out_features=768, bias=True)
            (LayerNorm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
      )
    )
  )
  (classifier): RobertaClassificationHead(
    (dense): Linear(in_features=768, out_features=768, bias=True)
    (dropout): Dropout(p=0.1, inplace=False)
    (out_proj): Linear(in_features=768, out_features=2, bias=True)
  )
)
```

)

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

```
roberta.embeddings.word_embeddings.weight requires_grad= True  
roberta.embeddings.position_embeddings.weight requires_grad= True  
roberta.embeddings.token_type_embeddings.weight requires_grad= True  
roberta.embeddings.LayerNorm.weight requires_grad= True  
roberta.embeddings.LayerNorm.bias requires_grad= True  
roberta.encoder.layer.0.attention.self.query.weight requires_grad= True  
roberta.encoder.layer.0.attention.self.query.bias requires_grad= True  
roberta.encoder.layer.0.attention.self.key.weight requires_grad= True  
roberta.encoder.layer.0.attention.self.key.bias requires_grad= True  
roberta.encoder.layer.0.attention.self.value.weight requires_grad= True  
roberta.encoder.layer.0.attention.self.value.bias requires_grad= True  
roberta.encoder.layer.0.attention.output.dense.weight requires_grad= True  
roberta.encoder.layer.0.attention.output.dense.bias requires_grad= True  
roberta.encoder.layer.0.attention.output.LayerNorm.weight requires_grad= True  
roberta.encoder.layer.0.attention.output.LayerNorm.bias requires_grad= True  
roberta.encoder.layer.0.intermediate.dense.weight requires_grad= True  
roberta.encoder.layer.0.intermediate.dense.bias requires_grad= True  
roberta.encoder.layer.0.output.dense.weight requires_grad= True  
roberta.encoder.layer.0.output.dense.bias requires_grad= True  
roberta.encoder.layer.0.output.LayerNorm.weight requires_grad= True  
roberta.encoder.layer.0.output.LayerNorm.bias requires_grad= True  
roberta.encoder.layer.1.attention.self.query.weight requires_grad= True  
roberta.encoder.layer.1.attention.self.query.bias requires_grad= True  
roberta.encoder.layer.1.attention.self.key.weight requires_grad= True  
roberta.encoder.layer.1.attention.self.key.bias requires_grad= True  
roberta.encoder.layer.1.attention.self.value.weight requires_grad= True  
roberta.encoder.layer.1.attention.self.value.bias requires_grad= True  
roberta.encoder.layer.1.attention.output.dense.weight requires_grad= True  
roberta.encoder.layer.1.attention.output.dense.bias requires_grad= True  
roberta.encoder.layer.1.attention.output.LayerNorm.weight requires_grad= True  
roberta.encoder.layer.1.attention.output.LayerNorm.bias requires_grad= True  
roberta.encoder.layer.1.intermediate.dense.weight requires_grad= True  
roberta.encoder.layer.1.intermediate.dense.bias requires_grad= True  
roberta.encoder.layer.1.output.dense.weight requires_grad= True  
roberta.encoder.layer.1.output.dense.bias requires_grad= True  
roberta.encoder.layer.1.output.LayerNorm.weight requires_grad= True  
roberta.encoder.layer.1.output.LayerNorm.bias requires_grad= True  
roberta.encoder.layer.2.attention.self.query.weight requires_grad= True  
roberta.encoder.layer.2.attention.self.query.bias requires_grad= True  
roberta.encoder.layer.2.attention.self.key.weight requires_grad= True  
roberta.encoder.layer.2.attention.self.key.bias requires_grad= True  
roberta.encoder.layer.2.attention.self.value.weight requires_grad= True  
roberta.encoder.layer.2.attention.self.value.bias requires_grad= True
```

[illegible]



[illegible]





```
1, 1, 1, 1, 1, 1, 1, 1])
```

```
[32]: layers_to_unfreeze = [
    "roberta.encoder.layer.11.attention.self.query.weight",
    "roberta.encoder.layer.11.attention.self.query.bias",
    "roberta.encoder.layer.11.attention.self.key.weight",
    "roberta.encoder.layer.11.attention.self.key.bias",
    "roberta.encoder.layer.11.attention.self.value.weight",
    "roberta.encoder.layer.11.attention.self.value.bias",
    "roberta.encoder.layer.11.attention.output.dense.weight",
    "roberta.encoder.layer.11.attention.output.dense.bias",
    "roberta.encoder.layer.11.attention.output.LayerNorm.weight",
    "roberta.encoder.layer.11.attention.output.LayerNorm.bias",
    "roberta.encoder.layer.11.intermediate.dense.weight",
    "roberta.encoder.layer.11.intermediate.dense.bias",
    "roberta.encoder.layer.11.output.dense.weight",
    "roberta.encoder.layer.11.output.dense.bias",
    "roberta.encoder.layer.11.output.LayerNorm.weight",
    "roberta.encoder.layer.11.output.LayerNorm.bias",
    "classifier.dense.weight",
    "classifier.dense.bias",
    "classifier.out_proj.weight",
    "classifier.out_proj.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)
```

```
RobertaConfig {
  "_attn_implementation_autoset": true,
  "architectures": [
    "RobertaForMaskedLM"
  ],
  "attention_probs_dropout_prob": 0.1,
```

```

    "bos_token_id": 0,
    "classifier_dropout": null,
    "eos_token_id": 2,
    "hidden_act": "gelu",
    "hidden_dropout_prob": 0.1,
    "hidden_size": 768,
    "initializer_range": 0.02,
    "intermediate_size": 3072,
    "layer_norm_eps": 1e-05,
    "max_position_embeddings": 514,
    "model_type": "roberta",
    "num_attention_heads": 12,
    "num_hidden_layers": 12,
    "pad_token_id": 1,
    "position_embedding_type": "absolute",
    "torch_dtype": "float32",
    "transformers_version": "4.50.3",
    "type_vocab_size": 1,
    "use_cache": true,
    "vocab_size": 50265
}

```

=====

```

num_parameters: 124647170
num_trainable_parameters: 7680002

```

=====

Experiment configuration used with this experiment:

```

model used: roberta-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: snc_morph_seq

```

```

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

```

```

roberta.embeddings.word_embeddings.weight requires_grad= False
roberta.embeddings.position_embeddings.weight requires_grad= False
roberta.embeddings.token_type_embeddings.weight requires_grad= False
roberta.embeddings.LayerNorm.weight requires_grad= False
roberta.embeddings.LayerNorm.bias requires_grad= False
roberta.encoder.layer.0.attention.self.query.weight requires_grad= False
roberta.encoder.layer.0.attention.self.query.bias requires_grad= False
roberta.encoder.layer.0.attention.self.key.weight requires_grad= False

```







```

roberta.encoder.layer.9.attention.self.key.bias requires_grad= False
roberta.encoder.layer.9.attention.self.value.weight requires_grad= False
roberta.encoder.layer.9.attention.self.value.bias requires_grad= False
roberta.encoder.layer.9.attention.output.dense.weight requires_grad= False
roberta.encoder.layer.9.attention.output.dense.bias requires_grad= False
roberta.encoder.layer.9.attention.output.LayerNorm.weight requires_grad= False
roberta.encoder.layer.9.attention.output.LayerNorm.bias requires_grad= False
roberta.encoder.layer.9.intermediate.dense.weight requires_grad= False
roberta.encoder.layer.9.intermediate.dense.bias requires_grad= False
roberta.encoder.layer.9.output.dense.weight requires_grad= False
roberta.encoder.layer.9.output.dense.bias requires_grad= False
roberta.encoder.layer.9.output.LayerNorm.weight requires_grad= False
roberta.encoder.layer.9.output.LayerNorm.bias requires_grad= False
roberta.encoder.layer.10.attention.self.query.weight requires_grad= False
roberta.encoder.layer.10.attention.self.query.bias requires_grad= False
roberta.encoder.layer.10.attention.self.key.weight requires_grad= False
roberta.encoder.layer.10.attention.self.key.bias requires_grad= False
roberta.encoder.layer.10.attention.self.value.weight requires_grad= False
roberta.encoder.layer.10.attention.self.value.bias requires_grad= False
roberta.encoder.layer.10.attention.output.dense.weight requires_grad= False
roberta.encoder.layer.10.attention.output.dense.bias requires_grad= False
roberta.encoder.layer.10.attention.output.LayerNorm.weight requires_grad= False
roberta.encoder.layer.10.attention.output.LayerNorm.bias requires_grad= False
roberta.encoder.layer.10.intermediate.dense.weight requires_grad= False
roberta.encoder.layer.10.intermediate.dense.bias requires_grad= False
roberta.encoder.layer.10.output.dense.weight requires_grad= False
roberta.encoder.layer.10.output.dense.bias requires_grad= False
roberta.encoder.layer.10.output.LayerNorm.weight requires_grad= False
roberta.encoder.layer.10.output.LayerNorm.bias requires_grad= False
roberta.encoder.layer.11.attention.self.query.weight requires_grad= True
roberta.encoder.layer.11.attention.self.query.bias requires_grad= True
roberta.encoder.layer.11.attention.self.key.weight requires_grad= True
roberta.encoder.layer.11.attention.self.key.bias requires_grad= True
roberta.encoder.layer.11.attention.self.value.weight requires_grad= True
roberta.encoder.layer.11.attention.self.value.bias requires_grad= True
roberta.encoder.layer.11.attention.output.dense.weight requires_grad= True
roberta.encoder.layer.11.attention.output.dense.bias requires_grad= True
roberta.encoder.layer.11.attention.output.LayerNorm.weight requires_grad= True
roberta.encoder.layer.11.attention.output.LayerNorm.bias requires_grad= True
roberta.encoder.layer.11.intermediate.dense.weight requires_grad= True
roberta.encoder.layer.11.intermediate.dense.bias requires_grad= True
roberta.encoder.layer.11.output.dense.weight requires_grad= True
roberta.encoder.layer.11.output.dense.bias requires_grad= True
roberta.encoder.layer.11.output.LayerNorm.weight requires_grad= True
roberta.encoder.layer.11.output.LayerNorm.bias requires_grad= True
classifier.dense.weight requires_grad= True
classifier.dense.bias requires_grad= True
classifier.out_proj.weight requires_grad= True

```

```
classifier.out_proj.bias requires_grad= True
```

```
[34]: # 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-20-0816cb2c5003>: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': 0.6924635171890259, 'eval_accuracy':
0.5151515151515151, 'eval_precision': 0.5151515151515151, 'eval_recall': 1.0,
'eval_f1': 0.68, 'eval_runtime': 4.9578, 'eval_samples_per_second': 19.969,
'eval_steps_per_second': 0.202, 'epoch': 1.0}
Test metrics: {'eval_loss': 0.6913613677024841, 'eval_accuracy':
0.5380434782608695, 'eval_precision': 0.5380434782608695, 'eval_recall': 1.0,
'eval_f1': 0.6996466431095406, 'eval_runtime': 5.0787,
'eval_samples_per_second': 36.23, 'eval_steps_per_second': 0.394, 'epoch': 1.0}
```

```
[35]: # 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\_roberta-  
base\_binary\_complexity\_20250411\_155935

<IPython.core.display.HTML object>

EXPERIMENT LOGGED TO:

/content/drive/MyDrive/266-final/results/experiment\_runs.txt

**0.2.5 snc bert-large-cased regularization\_weight\_decay = 0.5 learning\_rate = 5e-6  
size\_batch = 128 length\_max = 128 num\_epochs = 1**

```

[25]: # 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])

```





```

        (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)
)

```

```

[27]: 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

```

[illegible]











[illegible]



```

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

```

```

[28]: #####
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: snc_morph_seq

```

```

=====

```

```

num_trainable_parameters: 13647874

```

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

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

```
[30]: 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

```







[illegible]



[illegible]

[illegible]



```

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

```

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

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

```
[32]: # 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-20-0816cb2c5003>: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': 0.7925207614898682, 'eval_accuracy':  

0.494949494949495, 'eval_precision': 0.5333333333333333, 'eval_recall':  

0.1568627450980392, 'eval_f1': 0.24242424242424243, 'eval_runtime': 5.687,  

'eval_samples_per_second': 17.408, 'eval_steps_per_second': 0.176, 'epoch': 1.0}
Test metrics: {'eval_loss': 0.7933067679405212, 'eval_accuracy':  

0.4891304347826087, 'eval_precision': 0.5862068965517241, 'eval_recall':  

0.1717171717171717, 'eval_f1': 0.265625, 'eval_runtime': 5.6466,  

'eval_samples_per_second': 32.586, 'eval_steps_per_second': 0.354, 'epoch': 1.0}
```

```
[33]: # 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\_2025-04-11T09:02:02.177069-07:00

<IPython.core.display.HTML object>

EXPERIMENT LOGGED TO:

/content/drive/MyDrive/266-final/results/experiment\_runs.txt

**0.2.6 snc roberta-large regularization\_weight\_decay = 0.5 learning\_rate = 5e-6 size\_batch = 128 length\_max = 128 num\_epochs = 1**

```

[34]: # 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])
#####

```





```

        (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): RobertaSelfOutput(
        (dense): Linear(in_features=1024, out_features=1024, bias=True)
        (LayerNorm): LayerNorm((1024,), eps=1e-05,
elementwise_affine=True)
        (dropout): Dropout(p=0.1, inplace=False)
    )
)
(intermediate): RobertaIntermediate(
    (dense): Linear(in_features=1024, out_features=4096, bias=True)
    (intermediate_act_fn): GELUActivation()
)
(output): RobertaOutput(
    (dense): Linear(in_features=4096, out_features=1024, bias=True)
    (LayerNorm): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
    (dropout): Dropout(p=0.1, inplace=False)
)
)
)
)
)
)
(classifier): RobertaClassificationHead(
    (dense): Linear(in_features=1024, out_features=1024, bias=True)
    (dropout): Dropout(p=0.1, inplace=False)
    (out_proj): Linear(in_features=1024, out_features=2, bias=True)
)
)

```

```

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

```

```

roberta.embeddings.word_embeddings.weight requires_grad= True
roberta.embeddings.position_embeddings.weight requires_grad= True
roberta.embeddings.token_type_embeddings.weight requires_grad= True
roberta.embeddings.LayerNorm.weight requires_grad= True
roberta.embeddings.LayerNorm.bias requires_grad= True
roberta.encoder.layer.0.attention.self.query.weight requires_grad= True
roberta.encoder.layer.0.attention.self.query.bias requires_grad= True
roberta.encoder.layer.0.attention.self.key.weight requires_grad= True
roberta.encoder.layer.0.attention.self.key.bias requires_grad= True
roberta.encoder.layer.0.attention.self.value.weight requires_grad= True
roberta.encoder.layer.0.attention.self.value.bias requires_grad= True
roberta.encoder.layer.0.attention.output.dense.weight requires_grad= True
roberta.encoder.layer.0.attention.output.dense.bias requires_grad= True
roberta.encoder.layer.0.attention.output.LayerNorm.weight requires_grad= True

```

[illegible]



[illegible]



[illegible]



[illegible]



```

roberta.encoder.layer.21.attention.output.LayerNorm.bias requires_grad= True
roberta.encoder.layer.21.intermediate.dense.weight requires_grad= True
roberta.encoder.layer.21.intermediate.dense.bias requires_grad= True
roberta.encoder.layer.21.output.dense.weight requires_grad= True
roberta.encoder.layer.21.output.dense.bias requires_grad= True
roberta.encoder.layer.21.output.LayerNorm.weight requires_grad= True
roberta.encoder.layer.21.output.LayerNorm.bias requires_grad= True
roberta.encoder.layer.22.attention.self.query.weight requires_grad= True
roberta.encoder.layer.22.attention.self.query.bias requires_grad= True
roberta.encoder.layer.22.attention.self.key.weight requires_grad= True
roberta.encoder.layer.22.attention.self.key.bias requires_grad= True
roberta.encoder.layer.22.attention.self.value.weight requires_grad= True
roberta.encoder.layer.22.attention.self.value.bias requires_grad= True
roberta.encoder.layer.22.attention.output.dense.weight requires_grad= True
roberta.encoder.layer.22.attention.output.dense.bias requires_grad= True
roberta.encoder.layer.22.attention.output.LayerNorm.weight requires_grad= True
roberta.encoder.layer.22.attention.output.LayerNorm.bias requires_grad= True
roberta.encoder.layer.22.intermediate.dense.weight requires_grad= True
roberta.encoder.layer.22.intermediate.dense.bias requires_grad= True
roberta.encoder.layer.22.output.dense.weight requires_grad= True
roberta.encoder.layer.22.output.dense.bias requires_grad= True
roberta.encoder.layer.22.output.LayerNorm.weight requires_grad= True
roberta.encoder.layer.22.output.LayerNorm.bias requires_grad= True
roberta.encoder.layer.23.attention.self.query.weight requires_grad= True
roberta.encoder.layer.23.attention.self.query.bias requires_grad= True
roberta.encoder.layer.23.attention.self.key.weight requires_grad= True
roberta.encoder.layer.23.attention.self.key.bias requires_grad= True
roberta.encoder.layer.23.attention.self.value.weight requires_grad= True
roberta.encoder.layer.23.attention.self.value.bias requires_grad= True
roberta.encoder.layer.23.attention.output.dense.weight requires_grad= True
roberta.encoder.layer.23.attention.output.dense.bias requires_grad= True
roberta.encoder.layer.23.attention.output.LayerNorm.weight requires_grad= True
roberta.encoder.layer.23.attention.output.LayerNorm.bias requires_grad= True
roberta.encoder.layer.23.intermediate.dense.weight requires_grad= True
roberta.encoder.layer.23.intermediate.dense.bias requires_grad= True
roberta.encoder.layer.23.output.dense.weight requires_grad= True
roberta.encoder.layer.23.output.dense.bias requires_grad= True
roberta.encoder.layer.23.output.LayerNorm.weight requires_grad= True
roberta.encoder.layer.23.output.LayerNorm.bias requires_grad= True
classifier.dense.weight requires_grad= True
classifier.dense.bias requires_grad= True
classifier.out_proj.weight requires_grad= True
classifier.out_proj.bias requires_grad= True

```

```

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

```

[illegible]

```
[38]: #####
layers_to_unfreeze = [
    "roberta.encoder.layer.23.attention.self.query.weight",
    "roberta.encoder.layer.23.attention.self.query.bias",
    "roberta.encoder.layer.23.attention.self.key.weight",
    "roberta.encoder.layer.23.attention.self.key.bias",
    "roberta.encoder.layer.23.attention.self.value.weight",
    "roberta.encoder.layer.23.attention.self.value.bias",
    "roberta.encoder.layer.23.attention.output.dense.weight",
    "roberta.encoder.layer.23.attention.output.dense.bias",
    "roberta.encoder.layer.23.attention.output.LayerNorm.weight",
    "roberta.encoder.layer.23.attention.output.LayerNorm.bias",
    "roberta.encoder.layer.23.intermediate.dense.weight",
    "roberta.encoder.layer.23.intermediate.dense.bias",
    "roberta.encoder.layer.23.output.dense.weight",
    "roberta.encoder.layer.23.output.dense.bias",

```

```

        "roberta.encoder.layer.23.output.LayerNorm.weight",
        "roberta.encoder.layer.23.output.LayerNorm.bias",
        "classifier.dense.weight",
        "classifier.dense.bias",
        "classifier.out_proj.weight",
        "classifier.out_proj.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))

```

```

RobertaConfig {
  "_attn_implementation_autoset": true,
  "architectures": [
    "RobertaForMaskedLM"
  ],
  "attention_probs_dropout_prob": 0.1,
  "bos_token_id": 0,
  "classifier_dropout": null,
  "eos_token_id": 2,
  "hidden_act": "gelu",
  "hidden_dropout_prob": 0.1,
  "hidden_size": 1024,
  "initializer_range": 0.02,
  "intermediate_size": 4096,
  "layer_norm_eps": 1e-05,
  "max_position_embeddings": 514,
  "model_type": "roberta",
  "num_attention_heads": 16,
  "num_hidden_layers": 24,
  "pad_token_id": 1,

```

```

"position_embedding_type": "absolute",
"torch_dtype": "float32",
"transformers_version": "4.50.3",
"type_vocab_size": 1,
"use_cache": true,
"vocab_size": 50265
}

```

```

=====

```

```

num_parameters: 355361794
num_trainable_parameters: 13647874

```

```

=====

```

Experiment configuration used with this experiment:

```

model used: roberta-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: snc_morph_seq

```

```

=====

```

```

num_trainable_parameters: 13647874

```

```

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

```

```

roberta.embeddings.word_embeddings.weight requires_grad= False
roberta.embeddings.position_embeddings.weight requires_grad= False
roberta.embeddings.token_type_embeddings.weight requires_grad= False
roberta.embeddings.LayerNorm.weight requires_grad= False
roberta.embeddings.LayerNorm.bias requires_grad= False
roberta.encoder.layer.0.attention.self.query.weight requires_grad= False
roberta.encoder.layer.0.attention.self.query.bias requires_grad= False
roberta.encoder.layer.0.attention.self.key.weight requires_grad= False
roberta.encoder.layer.0.attention.self.key.bias requires_grad= False
roberta.encoder.layer.0.attention.self.value.weight requires_grad= False
roberta.encoder.layer.0.attention.self.value.bias requires_grad= False
roberta.encoder.layer.0.attention.output.dense.weight requires_grad= False
roberta.encoder.layer.0.attention.output.dense.bias requires_grad= False
roberta.encoder.layer.0.attention.output.LayerNorm.weight requires_grad= False
roberta.encoder.layer.0.attention.output.LayerNorm.bias requires_grad= False
roberta.encoder.layer.0.intermediate.dense.weight requires_grad= False
roberta.encoder.layer.0.intermediate.dense.bias requires_grad= False
roberta.encoder.layer.0.output.dense.weight requires_grad= False
roberta.encoder.layer.0.output.dense.bias requires_grad= False
roberta.encoder.layer.0.output.LayerNorm.weight requires_grad= False

```



[illegible]

[illegible]

[illegible]









```

roberta.encoder.layer.21.output.LayerNorm.bias requires_grad= False
roberta.encoder.layer.22.attention.self.query.weight requires_grad= False
roberta.encoder.layer.22.attention.self.query.bias requires_grad= False
roberta.encoder.layer.22.attention.self.key.weight requires_grad= False
roberta.encoder.layer.22.attention.self.key.bias requires_grad= False
roberta.encoder.layer.22.attention.self.value.weight requires_grad= False
roberta.encoder.layer.22.attention.self.value.bias requires_grad= False
roberta.encoder.layer.22.attention.output.dense.weight requires_grad= False
roberta.encoder.layer.22.attention.output.dense.bias requires_grad= False
roberta.encoder.layer.22.attention.output.LayerNorm.weight requires_grad= False
roberta.encoder.layer.22.attention.output.LayerNorm.bias requires_grad= False
roberta.encoder.layer.22.intermediate.dense.weight requires_grad= False
roberta.encoder.layer.22.intermediate.dense.bias requires_grad= False
roberta.encoder.layer.22.output.dense.weight requires_grad= False
roberta.encoder.layer.22.output.dense.bias requires_grad= False
roberta.encoder.layer.22.output.LayerNorm.weight requires_grad= False
roberta.encoder.layer.22.output.LayerNorm.bias requires_grad= False
roberta.encoder.layer.23.attention.self.query.weight requires_grad= True
roberta.encoder.layer.23.attention.self.query.bias requires_grad= True
roberta.encoder.layer.23.attention.self.key.weight requires_grad= True
roberta.encoder.layer.23.attention.self.key.bias requires_grad= True
roberta.encoder.layer.23.attention.self.value.weight requires_grad= True
roberta.encoder.layer.23.attention.self.value.bias requires_grad= True
roberta.encoder.layer.23.attention.output.dense.weight requires_grad= True
roberta.encoder.layer.23.attention.output.dense.bias requires_grad= True
roberta.encoder.layer.23.attention.output.LayerNorm.weight requires_grad= True
roberta.encoder.layer.23.attention.output.LayerNorm.bias requires_grad= True
roberta.encoder.layer.23.intermediate.dense.weight requires_grad= True
roberta.encoder.layer.23.intermediate.dense.bias requires_grad= True
roberta.encoder.layer.23.output.dense.weight requires_grad= True
roberta.encoder.layer.23.output.dense.bias requires_grad= True
roberta.encoder.layer.23.output.LayerNorm.weight requires_grad= True
roberta.encoder.layer.23.output.LayerNorm.bias requires_grad= True
classifier.dense.weight requires_grad= True
classifier.dense.bias requires_grad= True
classifier.out_proj.weight requires_grad= True
classifier.out_proj.bias requires_grad= True

```

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

```
[40]: Embedding(50265, 1024, padding_idx=1)
```

```
[41]: # 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-20-0816cb2c5003>: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': 0.6942740678787231, 'eval_accuracy':
0.5151515151515151, 'eval_precision': 0.5151515151515151, 'eval_recall': 1.0,
'eval_f1': 0.68, 'eval_runtime': 5.5378, 'eval_samples_per_second': 17.877,
'eval_steps_per_second': 0.181, 'epoch': 1.0}
Test metrics: {'eval_loss': 0.6895093321800232, 'eval_accuracy':
0.5380434782608695, 'eval_precision': 0.5380434782608695, 'eval_recall': 1.0,
'eval_f1': 0.6996466431095406, 'eval_runtime': 5.6698,
'eval_samples_per_second': 32.453, 'eval_steps_per_second': 0.353, 'epoch': 1.0}

```

```

[42]: # 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\_roberta-  
large\_binary\_complexity\_20250411\_160310

<IPython.core.display.HTML object>

EXPERIMENT LOGGED TO:

/content/drive/MyDrive/266-final/results/experiment\_runs.txt

**0.2.7 snc answerdotai/ModernBERT-base regularization\_weight\_decay = 0.5 learning\_rate = 5e-6 size\_batch = 128 length\_max = 128 num\_epochs = 1**

```

[43]: # 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(

```



```
tokenizer.json: 0%|          | 0.00/2.13M [00:00<?, ?B/s]
special_tokens_map.json: 0%|          | 0.00/694 [00:00<?, ?B/s]
config.json: 0%|          | 0.00/1.19k [00:00<?, ?B/s]
model.safetensors: 0%|          | 0.00/599M [00:00<?, ?B/s]
```

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': '2025-04-11 16:03:46'}
=====
```

```
[44]: 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)
)

```

```

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

```

```

model.embeddings.tok_embeddings.weight requires_grad= True
model.embeddings.norm.weight requires_grad= True
model.layers.0.attn.Wqkv.weight requires_grad= True
model.layers.0.attn.Wo.weight requires_grad= True
model.layers.0.mlp_norm.weight requires_grad= True
model.layers.0.mlp.Wi.weight requires_grad= True
model.layers.0.mlp.Wo.weight requires_grad= True
model.layers.1.attn_norm.weight requires_grad= True
model.layers.1.attn.Wqkv.weight requires_grad= True
model.layers.1.attn.Wo.weight requires_grad= True
model.layers.1.mlp_norm.weight requires_grad= True
model.layers.1.mlp.Wi.weight requires_grad= True
model.layers.1.mlp.Wo.weight requires_grad= True
model.layers.2.attn_norm.weight requires_grad= True
model.layers.2.attn.Wqkv.weight requires_grad= True
model.layers.2.attn.Wo.weight requires_grad= True
model.layers.2.mlp_norm.weight requires_grad= True
model.layers.2.mlp.Wi.weight requires_grad= True

```

model.layers.2.mlp.Wo.weight requires\_grad= True  
model.layers.3.attn\_norm.weight requires\_grad= True  
model.layers.3.attn.Wqkv.weight requires\_grad= True  
model.layers.3.attn.Wo.weight requires\_grad= True  
model.layers.3.mlp\_norm.weight requires\_grad= True  
model.layers.3.mlp.Wi.weight requires\_grad= True  
model.layers.3.mlp.Wo.weight requires\_grad= True  
model.layers.4.attn\_norm.weight requires\_grad= True  
model.layers.4.attn.Wqkv.weight requires\_grad= True  
model.layers.4.attn.Wo.weight requires\_grad= True  
model.layers.4.mlp\_norm.weight requires\_grad= True  
model.layers.4.mlp.Wi.weight requires\_grad= True  
model.layers.4.mlp.Wo.weight requires\_grad= True  
model.layers.5.attn\_norm.weight requires\_grad= True  
model.layers.5.attn.Wqkv.weight requires\_grad= True  
model.layers.5.attn.Wo.weight requires\_grad= True  
model.layers.5.mlp\_norm.weight requires\_grad= True  
model.layers.5.mlp.Wi.weight requires\_grad= True  
model.layers.5.mlp.Wo.weight requires\_grad= True  
model.layers.6.attn\_norm.weight requires\_grad= True  
model.layers.6.attn.Wqkv.weight requires\_grad= True  
model.layers.6.attn.Wo.weight requires\_grad= True  
model.layers.6.mlp\_norm.weight requires\_grad= True  
model.layers.6.mlp.Wi.weight requires\_grad= True  
model.layers.6.mlp.Wo.weight requires\_grad= True  
model.layers.7.attn\_norm.weight requires\_grad= True  
model.layers.7.attn.Wqkv.weight requires\_grad= True  
model.layers.7.attn.Wo.weight requires\_grad= True  
model.layers.7.mlp\_norm.weight requires\_grad= True  
model.layers.7.mlp.Wi.weight requires\_grad= True  
model.layers.7.mlp.Wo.weight requires\_grad= True  
model.layers.8.attn\_norm.weight requires\_grad= True  
model.layers.8.attn.Wqkv.weight requires\_grad= True  
model.layers.8.attn.Wo.weight requires\_grad= True  
model.layers.8.mlp\_norm.weight requires\_grad= True  
model.layers.8.mlp.Wi.weight requires\_grad= True  
model.layers.8.mlp.Wo.weight requires\_grad= True  
model.layers.9.attn\_norm.weight requires\_grad= True  
model.layers.9.attn.Wqkv.weight requires\_grad= True  
model.layers.9.attn.Wo.weight requires\_grad= True  
model.layers.9.mlp\_norm.weight requires\_grad= True  
model.layers.9.mlp.Wi.weight requires\_grad= True  
model.layers.9.mlp.Wo.weight requires\_grad= True  
model.layers.10.attn\_norm.weight requires\_grad= True  
model.layers.10.attn.Wqkv.weight requires\_grad= True  
model.layers.10.attn.Wo.weight requires\_grad= True  
model.layers.10.mlp\_norm.weight requires\_grad= True  
model.layers.10.mlp.Wi.weight requires\_grad= True





```

model.layers.18.mlp.Wo.weight requires_grad= True
model.layers.19.attn_norm.weight requires_grad= True
model.layers.19.attn.Wqkv.weight requires_grad= True
model.layers.19.attn.Wo.weight requires_grad= True
model.layers.19.mlp_norm.weight requires_grad= True
model.layers.19.mlp.Wi.weight requires_grad= True
model.layers.19.mlp.Wo.weight requires_grad= True
model.layers.20.attn_norm.weight requires_grad= True
model.layers.20.attn.Wqkv.weight requires_grad= True
model.layers.20.attn.Wo.weight requires_grad= True
model.layers.20.mlp_norm.weight requires_grad= True
model.layers.20.mlp.Wi.weight requires_grad= True
model.layers.20.mlp.Wo.weight requires_grad= True
model.layers.21.attn_norm.weight requires_grad= True
model.layers.21.attn.Wqkv.weight requires_grad= True
model.layers.21.attn.Wo.weight requires_grad= True
model.layers.21.mlp_norm.weight requires_grad= True
model.layers.21.mlp.Wi.weight requires_grad= True
model.layers.21.mlp.Wo.weight requires_grad= True
model.final_norm.weight requires_grad= True
head.dense.weight requires_grad= True
head.norm.weight requires_grad= True
classifier.weight requires_grad= True
classifier.bias requires_grad= True

```

```

[46]: # Inspect the attention_mask tensor for the first few samples
      for i in range(5):
          print(train_data_hf[i]['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, 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, 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])
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, 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, 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])
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, 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, 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])
tensor([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,

```

[illegible]

```
[47]: #####
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: snc_morph_seq
```

```
=====
```

```
num_trainable_parameters: 5607938
```

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

```
model.embeddings.tok_embeddings.weight requires_grad= False
```

```
model.embeddings.norm.weight requires_grad= False
```

```
model.layers.0.attn.Wqkv.weight requires_grad= False
```

```
model.layers.0.attn.Wo.weight requires_grad= False
```

```
model.layers.0.mlp_norm.weight requires_grad= False
```

```
model.layers.0.mlp.Wi.weight requires_grad= False
```

```
model.layers.0.mlp.Wo.weight requires_grad= False
```

```
model.layers.1.attn_norm.weight requires_grad= False
```

```
model.layers.1.attn.Wqkv.weight requires_grad= False
```

```
model.layers.1.attn.Wo.weight requires_grad= False
```

```
model.layers.1.mlp_norm.weight requires_grad= False
```

```
model.layers.1.mlp.Wi.weight requires_grad= False
```

```
model.layers.1.mlp.Wo.weight requires_grad= False
```

```
model.layers.2.attn_norm.weight requires_grad= False
```

```
model.layers.2.attn.Wqkv.weight requires_grad= False
```

```
model.layers.2.attn.Wo.weight requires_grad= False
```

```
model.layers.2.mlp_norm.weight requires_grad= False
```

```
model.layers.2.mlp.Wi.weight requires_grad= False
```

```
model.layers.2.mlp.Wo.weight requires_grad= False
```

```
model.layers.3.attn_norm.weight requires_grad= False
```

```
model.layers.3.attn.Wqkv.weight requires_grad= False
```

```
model.layers.3.attn.Wo.weight requires_grad= False
```

```
model.layers.3.mlp_norm.weight requires_grad= False
```

```
model.layers.3.mlp.Wi.weight requires_grad= False
```

```
model.layers.3.mlp.Wo.weight requires_grad= False
```

```
model.layers.4.attn_norm.weight requires_grad= False
```

```
model.layers.4.attn.Wqkv.weight requires_grad= False
```

```
model.layers.4.attn.Wo.weight requires_grad= False
```

```
model.layers.4.mlp_norm.weight requires_grad= False
```

```
model.layers.4.mlp.Wi.weight requires_grad= False
```

model.layers.4.mlp.Wo.weight requires\_grad= False  
model.layers.5.attn\_norm.weight requires\_grad= False  
model.layers.5.attn.Wqkv.weight requires\_grad= False  
model.layers.5.attn.Wo.weight requires\_grad= False  
model.layers.5.mlp\_norm.weight requires\_grad= False  
model.layers.5.mlp.Wi.weight requires\_grad= False  
model.layers.5.mlp.Wo.weight requires\_grad= False  
model.layers.6.attn\_norm.weight requires\_grad= False  
model.layers.6.attn.Wqkv.weight requires\_grad= False  
model.layers.6.attn.Wo.weight requires\_grad= False  
model.layers.6.mlp\_norm.weight requires\_grad= False  
model.layers.6.mlp.Wi.weight requires\_grad= False  
model.layers.6.mlp.Wo.weight requires\_grad= False  
model.layers.7.attn\_norm.weight requires\_grad= False  
model.layers.7.attn.Wqkv.weight requires\_grad= False  
model.layers.7.attn.Wo.weight requires\_grad= False  
model.layers.7.mlp\_norm.weight requires\_grad= False  
model.layers.7.mlp.Wi.weight requires\_grad= False  
model.layers.7.mlp.Wo.weight requires\_grad= False  
model.layers.8.attn\_norm.weight requires\_grad= False  
model.layers.8.attn.Wqkv.weight requires\_grad= False  
model.layers.8.attn.Wo.weight requires\_grad= False  
model.layers.8.mlp\_norm.weight requires\_grad= False  
model.layers.8.mlp.Wi.weight requires\_grad= False  
model.layers.8.mlp.Wo.weight requires\_grad= False  
model.layers.9.attn\_norm.weight requires\_grad= False  
model.layers.9.attn.Wqkv.weight requires\_grad= False  
model.layers.9.attn.Wo.weight requires\_grad= False  
model.layers.9.mlp\_norm.weight requires\_grad= False  
model.layers.9.mlp.Wi.weight requires\_grad= False  
model.layers.9.mlp.Wo.weight requires\_grad= False  
model.layers.10.attn\_norm.weight requires\_grad= False  
model.layers.10.attn.Wqkv.weight requires\_grad= False  
model.layers.10.attn.Wo.weight requires\_grad= False  
model.layers.10.mlp\_norm.weight requires\_grad= False  
model.layers.10.mlp.Wi.weight requires\_grad= False  
model.layers.10.mlp.Wo.weight requires\_grad= False  
model.layers.11.attn\_norm.weight requires\_grad= False  
model.layers.11.attn.Wqkv.weight requires\_grad= False  
model.layers.11.attn.Wo.weight requires\_grad= False  
model.layers.11.mlp\_norm.weight requires\_grad= False  
model.layers.11.mlp.Wi.weight requires\_grad= False  
model.layers.11.mlp.Wo.weight requires\_grad= False  
model.layers.12.attn\_norm.weight requires\_grad= False  
model.layers.12.attn.Wqkv.weight requires\_grad= False  
model.layers.12.attn.Wo.weight requires\_grad= False  
model.layers.12.mlp\_norm.weight requires\_grad= False  
model.layers.12.mlp.Wi.weight requires\_grad= False



```

model.layers.20.mlp.Wo.weight requires_grad= False
model.layers.21.attn_norm.weight requires_grad= True
model.layers.21.attn.Wqkv.weight requires_grad= True
model.layers.21.attn.Wo.weight requires_grad= True
model.layers.21.mlp_norm.weight requires_grad= True
model.layers.21.mlp.Wi.weight requires_grad= True
model.layers.21.mlp.Wo.weight requires_grad= True
model.final_norm.weight requires_grad= True
head.dense.weight requires_grad= True
head.norm.weight requires_grad= True
classifier.weight requires_grad= True
classifier.bias requires_grad= True

```

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

```
[50]: # 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-20-0816cb2c5003>:31: FutureWarning: `tokenizer` is deprecated and
will be removed in version 5.0.0 for `Trainer.__init__`. Use `processing_class`
instead.
    trainer = Trainer(
/usr/local/lib/python3.11/dist-packages/torch/_inductor/compile_fx.py:194:
UserWarning: TensorFloat32 tensor cores for float32 matrix multiplication
available but not enabled. Consider setting
`torch.set_float32_matmul_precision('high')` for better performance.
    warnings.warn(
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
Validation metrics: {'eval_loss': 1.9611859321594238, 'eval_accuracy':

```

```
0.5151515151515151, 'eval_precision': 0.5151515151515151, 'eval_recall': 1.0,
'eval_f1': 0.68, 'eval_runtime': 5.2572, 'eval_samples_per_second': 18.831,
'eval_steps_per_second': 0.19, 'epoch': 1.0}
Test metrics: {'eval_loss': 1.9127166271209717, 'eval_accuracy':
0.5380434782608695, 'eval_precision': 0.5380434782608695, 'eval_recall': 1.0,
'eval_f1': 0.6996466431095406, 'eval_runtime': 6.0082,
'eval_samples_per_second': 30.625, 'eval_steps_per_second': 0.333, 'epoch': 1.0}
```

```
[51]: # 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\_2025-04-11T09:04:25.944339-07:00

<IPython.core.display.HTML object>

EXPERIMENT LOGGED TO:  
/content/drive/MyDrive/266-final/results/experiment\_runs.txt



0.2.8 snc answerdotai/ModernBERT-large regularization\_weight\_decay = 0.5 learning\_rate = 5e-6 size\_batch = 128 length\_max = 128 num\_epochs = 1

```
[52]: # 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)
#####
# 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%| | 0/1517 [00:00<?, ? examples/s]

Map: 0%| | 0/99 [00:00<?, ? examples/s]

Map: 0%| | 0/184 [00:00<?, ? examples/s]

Datasets prepared. Sample from train\_data\_hf:

```

{'labels': tensor(0), 'input_ids': tensor([50281, 2598, 327, 253, 1735,
1388, 13, 672, 3419, 21743,
        66, 285, 10246, 547, 574, 1705, 342, 1270, 268, 23242,

```



```

        (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)
)

```

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

```
model.embeddings.tok_embeddings.weight requires_grad= True  
model.embeddings.norm.weight requires_grad= True  
model.layers.0.attn.Wqkv.weight requires_grad= True  
model.layers.0.attn.Wo.weight requires_grad= True  
model.layers.0.mlp_norm.weight requires_grad= True  
model.layers.0.mlp.Wi.weight requires_grad= True  
model.layers.0.mlp.Wo.weight requires_grad= True  
model.layers.1.attn_norm.weight requires_grad= True  
model.layers.1.attn.Wqkv.weight requires_grad= True  
model.layers.1.attn.Wo.weight requires_grad= True  
model.layers.1.mlp_norm.weight requires_grad= True  
model.layers.1.mlp.Wi.weight requires_grad= True  
model.layers.1.mlp.Wo.weight requires_grad= True  
model.layers.2.attn_norm.weight requires_grad= True  
model.layers.2.attn.Wqkv.weight requires_grad= True  
model.layers.2.attn.Wo.weight requires_grad= True  
model.layers.2.mlp_norm.weight requires_grad= True  
model.layers.2.mlp.Wi.weight requires_grad= True  
model.layers.2.mlp.Wo.weight requires_grad= True  
model.layers.3.attn_norm.weight requires_grad= True  
model.layers.3.attn.Wqkv.weight requires_grad= True  
model.layers.3.attn.Wo.weight requires_grad= True  
model.layers.3.mlp_norm.weight requires_grad= True  
model.layers.3.mlp.Wi.weight requires_grad= True  
model.layers.3.mlp.Wo.weight requires_grad= True  
model.layers.4.attn_norm.weight requires_grad= True  
model.layers.4.attn.Wqkv.weight requires_grad= True  
model.layers.4.attn.Wo.weight requires_grad= True  
model.layers.4.mlp_norm.weight requires_grad= True  
model.layers.4.mlp.Wi.weight requires_grad= True  
model.layers.4.mlp.Wo.weight requires_grad= True  
model.layers.5.attn_norm.weight requires_grad= True  
model.layers.5.attn.Wqkv.weight requires_grad= True  
model.layers.5.attn.Wo.weight requires_grad= True  
model.layers.5.mlp_norm.weight requires_grad= True  
model.layers.5.mlp.Wi.weight requires_grad= True  
model.layers.5.mlp.Wo.weight requires_grad= True  
model.layers.6.attn_norm.weight requires_grad= True  
model.layers.6.attn.Wqkv.weight requires_grad= True  
model.layers.6.attn.Wo.weight requires_grad= True  
model.layers.6.mlp_norm.weight requires_grad= True  
model.layers.6.mlp.Wi.weight requires_grad= True  
model.layers.6.mlp.Wo.weight requires_grad= True  
model.layers.7.attn_norm.weight requires_grad= True  
model.layers.7.attn.Wqkv.weight requires_grad= True
```

model.layers.7.attn.Wo.weight requires\_grad= True  
model.layers.7.mlp\_norm.weight requires\_grad= True  
model.layers.7.mlp.Wi.weight requires\_grad= True  
model.layers.7.mlp.Wo.weight requires\_grad= True  
model.layers.8.attn\_norm.weight requires\_grad= True  
model.layers.8.attn.Wqkv.weight requires\_grad= True  
model.layers.8.attn.Wo.weight requires\_grad= True  
model.layers.8.mlp\_norm.weight requires\_grad= True  
model.layers.8.mlp.Wi.weight requires\_grad= True  
model.layers.8.mlp.Wo.weight requires\_grad= True  
model.layers.9.attn\_norm.weight requires\_grad= True  
model.layers.9.attn.Wqkv.weight requires\_grad= True  
model.layers.9.attn.Wo.weight requires\_grad= True  
model.layers.9.mlp\_norm.weight requires\_grad= True  
model.layers.9.mlp.Wi.weight requires\_grad= True  
model.layers.9.mlp.Wo.weight requires\_grad= True  
model.layers.10.attn\_norm.weight requires\_grad= True  
model.layers.10.attn.Wqkv.weight requires\_grad= True  
model.layers.10.attn.Wo.weight requires\_grad= True  
model.layers.10.mlp\_norm.weight requires\_grad= True  
model.layers.10.mlp.Wi.weight requires\_grad= True  
model.layers.10.mlp.Wo.weight requires\_grad= True  
model.layers.11.attn\_norm.weight requires\_grad= True  
model.layers.11.attn.Wqkv.weight requires\_grad= True  
model.layers.11.attn.Wo.weight requires\_grad= True  
model.layers.11.mlp\_norm.weight requires\_grad= True  
model.layers.11.mlp.Wi.weight requires\_grad= True  
model.layers.11.mlp.Wo.weight requires\_grad= True  
model.layers.12.attn\_norm.weight requires\_grad= True  
model.layers.12.attn.Wqkv.weight requires\_grad= True  
model.layers.12.attn.Wo.weight requires\_grad= True  
model.layers.12.mlp\_norm.weight requires\_grad= True  
model.layers.12.mlp.Wi.weight requires\_grad= True  
model.layers.12.mlp.Wo.weight requires\_grad= True  
model.layers.13.attn\_norm.weight requires\_grad= True  
model.layers.13.attn.Wqkv.weight requires\_grad= True  
model.layers.13.attn.Wo.weight requires\_grad= True  
model.layers.13.mlp\_norm.weight requires\_grad= True  
model.layers.13.mlp.Wi.weight requires\_grad= True  
model.layers.13.mlp.Wo.weight requires\_grad= True  
model.layers.14.attn\_norm.weight requires\_grad= True  
model.layers.14.attn.Wqkv.weight requires\_grad= True  
model.layers.14.attn.Wo.weight requires\_grad= True  
model.layers.14.mlp\_norm.weight requires\_grad= True  
model.layers.14.mlp.Wi.weight requires\_grad= True  
model.layers.14.mlp.Wo.weight requires\_grad= True  
model.layers.15.attn\_norm.weight requires\_grad= True  
model.layers.15.attn.Wqkv.weight requires\_grad= True

model.layers.15.attn.Wo.weight requires\_grad= True  
model.layers.15.mlp\_norm.weight requires\_grad= True  
model.layers.15.mlp.Wi.weight requires\_grad= True  
model.layers.15.mlp.Wo.weight requires\_grad= True  
model.layers.16.attn\_norm.weight requires\_grad= True  
model.layers.16.attn.Wqkv.weight requires\_grad= True  
model.layers.16.attn.Wo.weight requires\_grad= True  
model.layers.16.mlp\_norm.weight requires\_grad= True  
model.layers.16.mlp.Wi.weight requires\_grad= True  
model.layers.16.mlp.Wo.weight requires\_grad= True  
model.layers.17.attn\_norm.weight requires\_grad= True  
model.layers.17.attn.Wqkv.weight requires\_grad= True  
model.layers.17.attn.Wo.weight requires\_grad= True  
model.layers.17.mlp\_norm.weight requires\_grad= True  
model.layers.17.mlp.Wi.weight requires\_grad= True  
model.layers.17.mlp.Wo.weight requires\_grad= True  
model.layers.18.attn\_norm.weight requires\_grad= True  
model.layers.18.attn.Wqkv.weight requires\_grad= True  
model.layers.18.attn.Wo.weight requires\_grad= True  
model.layers.18.mlp\_norm.weight requires\_grad= True  
model.layers.18.mlp.Wi.weight requires\_grad= True  
model.layers.18.mlp.Wo.weight requires\_grad= True  
model.layers.19.attn\_norm.weight requires\_grad= True  
model.layers.19.attn.Wqkv.weight requires\_grad= True  
model.layers.19.attn.Wo.weight requires\_grad= True  
model.layers.19.mlp\_norm.weight requires\_grad= True  
model.layers.19.mlp.Wi.weight requires\_grad= True  
model.layers.19.mlp.Wo.weight requires\_grad= True  
model.layers.20.attn\_norm.weight requires\_grad= True  
model.layers.20.attn.Wqkv.weight requires\_grad= True  
model.layers.20.attn.Wo.weight requires\_grad= True  
model.layers.20.mlp\_norm.weight requires\_grad= True  
model.layers.20.mlp.Wi.weight requires\_grad= True  
model.layers.20.mlp.Wo.weight requires\_grad= True  
model.layers.21.attn\_norm.weight requires\_grad= True  
model.layers.21.attn.Wqkv.weight requires\_grad= True  
model.layers.21.attn.Wo.weight requires\_grad= True  
model.layers.21.mlp\_norm.weight requires\_grad= True  
model.layers.21.mlp.Wi.weight requires\_grad= True  
model.layers.21.mlp.Wo.weight requires\_grad= True  
model.layers.22.attn\_norm.weight requires\_grad= True  
model.layers.22.attn.Wqkv.weight requires\_grad= True  
model.layers.22.attn.Wo.weight requires\_grad= True  
model.layers.22.mlp\_norm.weight requires\_grad= True  
model.layers.22.mlp.Wi.weight requires\_grad= True  
model.layers.22.mlp.Wo.weight requires\_grad= True  
model.layers.23.attn\_norm.weight requires\_grad= True  
model.layers.23.attn.Wqkv.weight requires\_grad= True

```

model.layers.23.attn.Wo.weight requires_grad= True
model.layers.23.mlp_norm.weight requires_grad= True
model.layers.23.mlp.Wi.weight requires_grad= True
model.layers.23.mlp.Wo.weight requires_grad= True
model.layers.24.attn_norm.weight requires_grad= True
model.layers.24.attn.Wqkv.weight requires_grad= True
model.layers.24.attn.Wo.weight requires_grad= True
model.layers.24.mlp_norm.weight requires_grad= True
model.layers.24.mlp.Wi.weight requires_grad= True
model.layers.24.mlp.Wo.weight requires_grad= True
model.layers.25.attn_norm.weight requires_grad= True
model.layers.25.attn.Wqkv.weight requires_grad= True
model.layers.25.attn.Wo.weight requires_grad= True
model.layers.25.mlp_norm.weight requires_grad= True
model.layers.25.mlp.Wi.weight requires_grad= True
model.layers.25.mlp.Wo.weight requires_grad= True
model.layers.26.attn_norm.weight requires_grad= True
model.layers.26.attn.Wqkv.weight requires_grad= True
model.layers.26.attn.Wo.weight requires_grad= True
model.layers.26.mlp_norm.weight requires_grad= True
model.layers.26.mlp.Wi.weight requires_grad= True
model.layers.26.mlp.Wo.weight requires_grad= True
model.layers.27.attn_norm.weight requires_grad= True
model.layers.27.attn.Wqkv.weight requires_grad= True
model.layers.27.attn.Wo.weight requires_grad= True
model.layers.27.mlp_norm.weight requires_grad= True
model.layers.27.mlp.Wi.weight requires_grad= True
model.layers.27.mlp.Wo.weight requires_grad= True
model.final_norm.weight requires_grad= True
head.dense.weight requires_grad= True
head.norm.weight requires_grad= True
classifier.weight requires_grad= True
classifier.bias requires_grad= True

```

```

[55]: # Inspect the attention_mask tensor for the first few samples
      for i in range(5):
          print(train_data_hf[i]['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, 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, 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])
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, 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, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1])

```



```
#####
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)
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: snc\_morph\_seq

```

=====

```

```

num_trainable_parameters: 13309954

```

```

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

```

```

model.embeddings.tok_embeddings.weight requires_grad= False
model.embeddings.norm.weight requires_grad= False
model.layers.0.attn.Wqkv.weight requires_grad= False
model.layers.0.attn.Wo.weight requires_grad= False
model.layers.0.mlp_norm.weight requires_grad= False
model.layers.0.mlp.Wi.weight requires_grad= False
model.layers.0.mlp.Wo.weight requires_grad= False
model.layers.1.attn_norm.weight requires_grad= False
model.layers.1.attn.Wqkv.weight requires_grad= False
model.layers.1.attn.Wo.weight requires_grad= False
model.layers.1.mlp_norm.weight requires_grad= False
model.layers.1.mlp.Wi.weight requires_grad= False
model.layers.1.mlp.Wo.weight requires_grad= False
model.layers.2.attn_norm.weight requires_grad= False
model.layers.2.attn.Wqkv.weight requires_grad= False
model.layers.2.attn.Wo.weight requires_grad= False
model.layers.2.mlp_norm.weight requires_grad= False
model.layers.2.mlp.Wi.weight requires_grad= False
model.layers.2.mlp.Wo.weight requires_grad= False

```

model.layers.3.attn\_norm.weight requires\_grad= False  
model.layers.3.attn.Wqkv.weight requires\_grad= False  
model.layers.3.attn.Wo.weight requires\_grad= False  
model.layers.3.mlp\_norm.weight requires\_grad= False  
model.layers.3.mlp.Wi.weight requires\_grad= False  
model.layers.3.mlp.Wo.weight requires\_grad= False  
model.layers.4.attn\_norm.weight requires\_grad= False  
model.layers.4.attn.Wqkv.weight requires\_grad= False  
model.layers.4.attn.Wo.weight requires\_grad= False  
model.layers.4.mlp\_norm.weight requires\_grad= False  
model.layers.4.mlp.Wi.weight requires\_grad= False  
model.layers.4.mlp.Wo.weight requires\_grad= False  
model.layers.5.attn\_norm.weight requires\_grad= False  
model.layers.5.attn.Wqkv.weight requires\_grad= False  
model.layers.5.attn.Wo.weight requires\_grad= False  
model.layers.5.mlp\_norm.weight requires\_grad= False  
model.layers.5.mlp.Wi.weight requires\_grad= False  
model.layers.5.mlp.Wo.weight requires\_grad= False  
model.layers.6.attn\_norm.weight requires\_grad= False  
model.layers.6.attn.Wqkv.weight requires\_grad= False  
model.layers.6.attn.Wo.weight requires\_grad= False  
model.layers.6.mlp\_norm.weight requires\_grad= False  
model.layers.6.mlp.Wi.weight requires\_grad= False  
model.layers.6.mlp.Wo.weight requires\_grad= False  
model.layers.7.attn\_norm.weight requires\_grad= False  
model.layers.7.attn.Wqkv.weight requires\_grad= False  
model.layers.7.attn.Wo.weight requires\_grad= False  
model.layers.7.mlp\_norm.weight requires\_grad= False  
model.layers.7.mlp.Wi.weight requires\_grad= False  
model.layers.7.mlp.Wo.weight requires\_grad= False  
model.layers.8.attn\_norm.weight requires\_grad= False  
model.layers.8.attn.Wqkv.weight requires\_grad= False  
model.layers.8.attn.Wo.weight requires\_grad= False  
model.layers.8.mlp\_norm.weight requires\_grad= False  
model.layers.8.mlp.Wi.weight requires\_grad= False  
model.layers.8.mlp.Wo.weight requires\_grad= False  
model.layers.9.attn\_norm.weight requires\_grad= False  
model.layers.9.attn.Wqkv.weight requires\_grad= False  
model.layers.9.attn.Wo.weight requires\_grad= False  
model.layers.9.mlp\_norm.weight requires\_grad= False  
model.layers.9.mlp.Wi.weight requires\_grad= False  
model.layers.9.mlp.Wo.weight requires\_grad= False  
model.layers.10.attn\_norm.weight requires\_grad= False  
model.layers.10.attn.Wqkv.weight requires\_grad= False  
model.layers.10.attn.Wo.weight requires\_grad= False  
model.layers.10.mlp\_norm.weight requires\_grad= False  
model.layers.10.mlp.Wi.weight requires\_grad= False  
model.layers.10.mlp.Wo.weight requires\_grad= False





```

model.layers.27.attn_norm.weight requires_grad= True
model.layers.27.attn.Wqkv.weight requires_grad= True
model.layers.27.attn.Wo.weight requires_grad= True
model.layers.27.mlp_norm.weight requires_grad= True
model.layers.27.mlp.Wi.weight requires_grad= True
model.layers.27.mlp.Wo.weight requires_grad= True
model.final_norm.weight requires_grad= True
head.dense.weight requires_grad= True
head.norm.weight requires_grad= True
classifier.weight requires_grad= True
classifier.bias requires_grad= True

```

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

```

[59]: # 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-20-0816cb2c5003>: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>

W0411 16:05:14.614000 2541 torch/_dynamo/convert_frame.py:906] [1/8]
torch._dynamo hit config.cache_size_limit (8)
W0411 16:05:14.614000 2541 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)
W0411 16:05:14.614000 2541 torch/_dynamo/convert_frame.py:906] [1/8]      last
reason: 1/0: GLOBAL_STATE changed: grad_mode
W0411 16:05:14.614000 2541 torch/_dynamo/convert_frame.py:906] [1/8] To log all

```

recompilation reasons, use TORCH\_LOGS="recompiles".  
W0411 16:05:14.614000 2541 torch/\_dynamo/convert\_frame.py:906] [1/8] To diagnose  
recompilation issues, see  
[https://pytorch.org/docs/main/torch.compiler\\_troubleshooting.html](https://pytorch.org/docs/main/torch.compiler_troubleshooting.html).

<IPython.core.display.HTML object>

Validation metrics: {'eval\_loss': 0.6933701634407043, 'eval\_accuracy':  
0.5353535353535354, 'eval\_precision': 0.5531914893617021, 'eval\_recall':  
0.5098039215686274, 'eval\_f1': 0.5306122448979592, 'eval\_runtime': 5.5474,  
'eval\_samples\_per\_second': 17.846, 'eval\_steps\_per\_second': 0.18, 'epoch': 1.0}  
Test metrics: {'eval\_loss': 0.6932283043861389, 'eval\_accuracy':  
0.5054347826086957, 'eval\_precision': 0.55, 'eval\_recall': 0.4444444444444444,  
'eval\_f1': 0.49162011173184356, 'eval\_runtime': 5.9803,  
'eval\_samples\_per\_second': 30.768, 'eval\_steps\_per\_second': 0.334, 'epoch': 1.0}

```
[60]: # 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\_20250411\_160533



<IPython.core.display.HTML object>

EXPERIMENT LOGGED TO:

/content/drive/MyDrive/266-final/results/experiment\_runs.txt

0.2.9 snc microsoft/deberta-v3-base regularization\_weight\_decay = 0.5 learning\_rate = 5e-6 size\_batch = 128 length\_max = 128 num\_epochs = 1

```
[61]: # 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
# named_model = "answerdotai/ModernBERT-large" # modern bert
named_model = "microsoft/deberta-v3-base" # deberta
#####
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="microsoft/deberta-v3-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%| | 0/1517 [00:00<?, ? examples/s]

Map: 0%| | 0/99 [00:00<?, ? examples/s]

```
Map: 0%|          | 0/184 [00:00<?, ? examples/s]
```

Datasets prepared. Sample from train\_data\_hf:

[illegible]

Loading from Hugging Face model: microsoft/deberta-v3-base

```
tokenizer_config.json: 0%|          | 0.00/52.0 [00:00<?, ?B/s]
```

```
config.json: 0%|          | 0.00/579 [00:00<?, ?B/s]
```

```
spm.model: 0%|          | 0.00/2.46M [00:00<?, ?B/s]
```

```
/usr/local/lib/python3.11/dist-
```

packages/transformers/convert\_slow\_tokenizer.py:559: UserWarning: The sentencepiece tokenizer that you are converting to a fast tokenizer uses the byte fallback option which is not implemented in the fast tokenizers. In practice this means that the fast version of the tokenizer can produce unknown tokens whereas the sentencepiece version would have converted these unknown tokens into a sequence of byte tokens matching the original piece of text.

```
warnings.warn(
```

```
pytorch_model.bin: 0%|          | 0.00/371M [00:00<?, ?B/s]
```

Some weights of DebertaV2ForSequenceClassification were not initialized from the model checkpoint at microsoft/deberta-v3-base and are newly initialized:

```
['classifier.bias', 'classifier.weight', 'pooler.dense.bias',  
'pooler.dense.weight']
```

You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.

=====

microsoft/deberta-v3-base :

```

=====
num_parameters: 184423682
num_trainable_parameters at load: 184423682
=====
model lineage: {'type': 'huggingface_hub', 'path': 'microsoft/deberta-v3-base',
'timestamp': '2025-04-11 16:06:09'}
=====

```

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

```

DebertaV2ForSequenceClassification(
  (deberta): DebertaV2Model(
    (embeddings): DebertaV2Embeddings(
      (word_embeddings): Embedding(128100, 768, padding_idx=0)
      (LayerNorm): LayerNorm((768,), eps=1e-07, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (encoder): DebertaV2Encoder(
      (layer): ModuleList(
        (0-11): 12 x DebertaV2Layer(
          (attention): DebertaV2Attention(
            (self): DisentangledSelfAttention(
              (query_proj): Linear(in_features=768, out_features=768, bias=True)
              (key_proj): Linear(in_features=768, out_features=768, bias=True)
              (value_proj): Linear(in_features=768, out_features=768, bias=True)
              (pos_dropout): Dropout(p=0.1, inplace=False)
              (dropout): Dropout(p=0.1, inplace=False)
            )
            (output): DebertaV2SelfOutput(
              (dense): Linear(in_features=768, out_features=768, bias=True)
              (LayerNorm): LayerNorm((768,), eps=1e-07, elementwise_affine=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
          )
          (intermediate): DebertaV2Intermediate(
            (dense): Linear(in_features=768, out_features=3072, bias=True)
            (intermediate_act_fn): GELUActivation()
          )
          (output): DebertaV2Output(
            (dense): Linear(in_features=3072, out_features=768, bias=True)
            (LayerNorm): LayerNorm((768,), eps=1e-07, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
      )
      (rel_embeddings): Embedding(512, 768)
      (LayerNorm): LayerNorm((768,), eps=1e-07, elementwise_affine=True)
    )
  )
)

```

```

)
(pooler): ContextPooler(
  (dense): Linear(in_features=768, out_features=768, bias=True)
  (dropout): Dropout(p=0, inplace=False)
)
(classifier): Linear(in_features=768, out_features=2, bias=True)
(dropout): Dropout(p=0.1, inplace=False)
)

```

```

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

```

```

deberta.embeddings.word_embeddings.weight requires_grad= True
deberta.embeddings.LayerNorm.weight requires_grad= True
deberta.embeddings.LayerNorm.bias requires_grad= True
deberta.encoder.layer.0.attention.self.query_proj.weight requires_grad= True
deberta.encoder.layer.0.attention.self.query_proj.bias requires_grad= True
deberta.encoder.layer.0.attention.self.key_proj.weight requires_grad= True
deberta.encoder.layer.0.attention.self.key_proj.bias requires_grad= True
deberta.encoder.layer.0.attention.self.value_proj.weight requires_grad= True
deberta.encoder.layer.0.attention.self.value_proj.bias requires_grad= True
deberta.encoder.layer.0.attention.output.dense.weight requires_grad= True
deberta.encoder.layer.0.attention.output.dense.bias requires_grad= True
deberta.encoder.layer.0.attention.output.LayerNorm.weight requires_grad= True
deberta.encoder.layer.0.attention.output.LayerNorm.bias requires_grad= True
deberta.encoder.layer.0.intermediate.dense.weight requires_grad= True
deberta.encoder.layer.0.intermediate.dense.bias requires_grad= True
deberta.encoder.layer.0.output.dense.weight requires_grad= True
deberta.encoder.layer.0.output.dense.bias requires_grad= True
deberta.encoder.layer.0.output.LayerNorm.weight requires_grad= True
deberta.encoder.layer.0.output.LayerNorm.bias requires_grad= True
deberta.encoder.layer.1.attention.self.query_proj.weight requires_grad= True
deberta.encoder.layer.1.attention.self.query_proj.bias requires_grad= True
deberta.encoder.layer.1.attention.self.key_proj.weight requires_grad= True
deberta.encoder.layer.1.attention.self.key_proj.bias requires_grad= True
deberta.encoder.layer.1.attention.self.value_proj.weight requires_grad= True
deberta.encoder.layer.1.attention.self.value_proj.bias requires_grad= True
deberta.encoder.layer.1.attention.output.dense.weight requires_grad= True
deberta.encoder.layer.1.attention.output.dense.bias requires_grad= True
deberta.encoder.layer.1.attention.output.LayerNorm.weight requires_grad= True
deberta.encoder.layer.1.attention.output.LayerNorm.bias requires_grad= True
deberta.encoder.layer.1.intermediate.dense.weight requires_grad= True
deberta.encoder.layer.1.intermediate.dense.bias requires_grad= True
deberta.encoder.layer.1.output.dense.weight requires_grad= True
deberta.encoder.layer.1.output.dense.bias requires_grad= True
deberta.encoder.layer.1.output.LayerNorm.weight requires_grad= True
deberta.encoder.layer.1.output.LayerNorm.bias requires_grad= True
deberta.encoder.layer.2.attention.self.query_proj.weight requires_grad= True

```





deberta.encoder.layer.8.attention.self.query\_proj.bias requires\_grad= True  
deberta.encoder.layer.8.attention.self.key\_proj.weight requires\_grad= True  
deberta.encoder.layer.8.attention.self.key\_proj.bias requires\_grad= True  
deberta.encoder.layer.8.attention.self.value\_proj.weight requires\_grad= True  
deberta.encoder.layer.8.attention.self.value\_proj.bias requires\_grad= True  
deberta.encoder.layer.8.attention.output.dense.weight requires\_grad= True  
deberta.encoder.layer.8.attention.output.dense.bias requires\_grad= True  
deberta.encoder.layer.8.attention.output.LayerNorm.weight requires\_grad= True  
deberta.encoder.layer.8.attention.output.LayerNorm.bias requires\_grad= True  
deberta.encoder.layer.8.intermediate.dense.weight requires\_grad= True  
deberta.encoder.layer.8.intermediate.dense.bias requires\_grad= True  
deberta.encoder.layer.8.output.dense.weight requires\_grad= True  
deberta.encoder.layer.8.output.dense.bias requires\_grad= True  
deberta.encoder.layer.8.output.LayerNorm.weight requires\_grad= True  
deberta.encoder.layer.8.output.LayerNorm.bias requires\_grad= True  
deberta.encoder.layer.9.attention.self.query\_proj.weight requires\_grad= True  
deberta.encoder.layer.9.attention.self.query\_proj.bias requires\_grad= True  
deberta.encoder.layer.9.attention.self.key\_proj.weight requires\_grad= True  
deberta.encoder.layer.9.attention.self.key\_proj.bias requires\_grad= True  
deberta.encoder.layer.9.attention.self.value\_proj.weight requires\_grad= True  
deberta.encoder.layer.9.attention.self.value\_proj.bias requires\_grad= True  
deberta.encoder.layer.9.attention.output.dense.weight requires\_grad= True  
deberta.encoder.layer.9.attention.output.dense.bias requires\_grad= True  
deberta.encoder.layer.9.attention.output.LayerNorm.weight requires\_grad= True  
deberta.encoder.layer.9.attention.output.LayerNorm.bias requires\_grad= True  
deberta.encoder.layer.9.intermediate.dense.weight requires\_grad= True  
deberta.encoder.layer.9.intermediate.dense.bias requires\_grad= True  
deberta.encoder.layer.9.output.dense.weight requires\_grad= True  
deberta.encoder.layer.9.output.dense.bias requires\_grad= True  
deberta.encoder.layer.9.output.LayerNorm.weight requires\_grad= True  
deberta.encoder.layer.9.output.LayerNorm.bias requires\_grad= True  
deberta.encoder.layer.10.attention.self.query\_proj.weight requires\_grad= True  
deberta.encoder.layer.10.attention.self.query\_proj.bias requires\_grad= True  
deberta.encoder.layer.10.attention.self.key\_proj.weight requires\_grad= True  
deberta.encoder.layer.10.attention.self.key\_proj.bias requires\_grad= True  
deberta.encoder.layer.10.attention.self.value\_proj.weight requires\_grad= True  
deberta.encoder.layer.10.attention.self.value\_proj.bias requires\_grad= True  
deberta.encoder.layer.10.attention.output.dense.weight requires\_grad= True  
deberta.encoder.layer.10.attention.output.dense.bias requires\_grad= True  
deberta.encoder.layer.10.attention.output.LayerNorm.weight requires\_grad= True  
deberta.encoder.layer.10.attention.output.LayerNorm.bias requires\_grad= True  
deberta.encoder.layer.10.intermediate.dense.weight requires\_grad= True  
deberta.encoder.layer.10.intermediate.dense.bias requires\_grad= True  
deberta.encoder.layer.10.output.dense.weight requires\_grad= True  
deberta.encoder.layer.10.output.dense.bias requires\_grad= True  
deberta.encoder.layer.10.output.LayerNorm.weight requires\_grad= True  
deberta.encoder.layer.10.output.LayerNorm.bias requires\_grad= True  
deberta.encoder.layer.11.attention.self.query\_proj.weight requires\_grad= True



```

deberta.encoder.layer.11.attention.self.query_proj.bias requires_grad= True
deberta.encoder.layer.11.attention.self.key_proj.weight requires_grad= True
deberta.encoder.layer.11.attention.self.key_proj.bias requires_grad= True
deberta.encoder.layer.11.attention.self.value_proj.weight requires_grad= True
deberta.encoder.layer.11.attention.self.value_proj.bias requires_grad= True
deberta.encoder.layer.11.attention.output.dense.weight requires_grad= True
deberta.encoder.layer.11.attention.output.dense.bias requires_grad= True
deberta.encoder.layer.11.attention.output.LayerNorm.weight requires_grad= True
deberta.encoder.layer.11.attention.output.LayerNorm.bias requires_grad= True
deberta.encoder.layer.11.intermediate.dense.weight requires_grad= True
deberta.encoder.layer.11.intermediate.dense.bias requires_grad= True
deberta.encoder.layer.11.output.dense.weight requires_grad= True
deberta.encoder.layer.11.output.dense.bias requires_grad= True
deberta.encoder.layer.11.output.LayerNorm.weight requires_grad= True
deberta.encoder.layer.11.output.LayerNorm.bias requires_grad= True
deberta.encoder.rel_embeddings.weight requires_grad= True
deberta.encoder.LayerNorm.weight requires_grad= True
deberta.encoder.LayerNorm.bias requires_grad= True
pooler.dense.weight requires_grad= True
pooler.dense.bias requires_grad= True
classifier.weight requires_grad= True
classifier.bias requires_grad= True

```

```

[64]: #####
layers_to_unfreeze = [
    "deberta.encoder.layer.11.attention.self.query_proj.weight",
    "deberta.encoder.layer.11.attention.self.query_proj.bias",
    "deberta.encoder.layer.11.attention.self.key_proj.weight",
    "deberta.encoder.layer.11.attention.self.key_proj.bias",
    "deberta.encoder.layer.11.attention.self.value_proj.weight",
    "deberta.encoder.layer.11.attention.self.value_proj.bias",
    "deberta.encoder.layer.11.attention.output.dense.weight",
    "deberta.encoder.layer.11.attention.output.dense.bias",
    "deberta.encoder.layer.11.attention.output.LayerNorm.weight",
    "deberta.encoder.layer.11.attention.output.LayerNorm.bias",
    "deberta.encoder.layer.11.intermediate.dense.weight",
    "deberta.encoder.layer.11.intermediate.dense.bias",
    "deberta.encoder.layer.11.output.dense.weight",
    "deberta.encoder.layer.11.output.dense.bias",
    "deberta.encoder.layer.11.output.LayerNorm.weight",
    "deberta.encoder.layer.11.output.LayerNorm.bias",
    "deberta.encoder.rel_embeddings.weight",
    "deberta.encoder.LayerNorm.weight",
    "deberta.encoder.LayerNorm.bias",
    "pooler.dense.weight",
    "pooler.dense.bias",
    "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))

```

```

DebertaV2Config {
  "_attn_implementation_autoset": true,
  "attention_probs_dropout_prob": 0.1,
  "hidden_act": "gelu",
  "hidden_dropout_prob": 0.1,
  "hidden_size": 768,
  "initializer_range": 0.02,
  "intermediate_size": 3072,
  "layer_norm_eps": 1e-07,
  "legacy": true,
  "max_position_embeddings": 512,
  "max_relative_positions": -1,
  "model_type": "deberta-v2",
  "norm_rel_ebd": "layer_norm",
  "num_attention_heads": 12,
  "num_hidden_layers": 12,
  "pad_token_id": 0,
  "pooler_dropout": 0,
  "pooler_hidden_act": "gelu",
  "pooler_hidden_size": 768,
  "pos_att_type": [
    "p2c",
    "c2p"
  ],
  "position_biased_input": false,

```

```

    "position_buckets": 256,
    "relative_attention": true,
    "share_att_key": true,
    "torch_dtype": "float32",
    "transformers_version": "4.50.3",
    "type_vocab_size": 0,
    "vocab_size": 128100
}

```

```

=====

```

```

num_parameters: 184423682
num_trainable_parameters: 8074754

```

```

=====

```

Experiment configuration used with this experiment:

model used: microsoft/deberta-v3-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: snc\_morph\_seq

```

=====

```

```

num_trainable_parameters: 8074754

```

```

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

```

```

deberta.embeddings.word_embeddings.weight requires_grad= False
deberta.embeddings.LayerNorm.weight requires_grad= False
deberta.embeddings.LayerNorm.bias requires_grad= False
deberta.encoder.layer.0.attention.self.query_proj.weight requires_grad= False
deberta.encoder.layer.0.attention.self.query_proj.bias requires_grad= False
deberta.encoder.layer.0.attention.self.key_proj.weight requires_grad= False
deberta.encoder.layer.0.attention.self.key_proj.bias requires_grad= False
deberta.encoder.layer.0.attention.self.value_proj.weight requires_grad= False
deberta.encoder.layer.0.attention.self.value_proj.bias requires_grad= False
deberta.encoder.layer.0.attention.output.dense.weight requires_grad= False
deberta.encoder.layer.0.attention.output.dense.bias requires_grad= False
deberta.encoder.layer.0.attention.output.LayerNorm.weight requires_grad= False
deberta.encoder.layer.0.attention.output.LayerNorm.bias requires_grad= False
deberta.encoder.layer.0.intermediate.dense.weight requires_grad= False
deberta.encoder.layer.0.intermediate.dense.bias requires_grad= False
deberta.encoder.layer.0.output.dense.weight requires_grad= False
deberta.encoder.layer.0.output.dense.bias requires_grad= False
deberta.encoder.layer.0.output.LayerNorm.weight requires_grad= False
deberta.encoder.layer.0.output.LayerNorm.bias requires_grad= False

```





[illegible]

```

deberta.encoder.layer.10.attention.self.query_proj.weight requires_grad= False
deberta.encoder.layer.10.attention.self.query_proj.bias requires_grad= False
deberta.encoder.layer.10.attention.self.key_proj.weight requires_grad= False
deberta.encoder.layer.10.attention.self.key_proj.bias requires_grad= False
deberta.encoder.layer.10.attention.self.value_proj.weight requires_grad= False
deberta.encoder.layer.10.attention.self.value_proj.bias requires_grad= False
deberta.encoder.layer.10.attention.output.dense.weight requires_grad= False
deberta.encoder.layer.10.attention.output.dense.bias requires_grad= False
deberta.encoder.layer.10.attention.output.LayerNorm.weight requires_grad= False
deberta.encoder.layer.10.attention.output.LayerNorm.bias requires_grad= False
deberta.encoder.layer.10.intermediate.dense.weight requires_grad= False
deberta.encoder.layer.10.intermediate.dense.bias requires_grad= False
deberta.encoder.layer.10.output.dense.weight requires_grad= False
deberta.encoder.layer.10.output.dense.bias requires_grad= False
deberta.encoder.layer.10.output.LayerNorm.weight requires_grad= False
deberta.encoder.layer.10.output.LayerNorm.bias requires_grad= False
deberta.encoder.layer.11.attention.self.query_proj.weight requires_grad= True
deberta.encoder.layer.11.attention.self.query_proj.bias requires_grad= True
deberta.encoder.layer.11.attention.self.key_proj.weight requires_grad= True
deberta.encoder.layer.11.attention.self.key_proj.bias requires_grad= True
deberta.encoder.layer.11.attention.self.value_proj.weight requires_grad= True
deberta.encoder.layer.11.attention.self.value_proj.bias requires_grad= True
deberta.encoder.layer.11.attention.output.dense.weight requires_grad= True
deberta.encoder.layer.11.attention.output.dense.bias requires_grad= True
deberta.encoder.layer.11.attention.output.LayerNorm.weight requires_grad= True
deberta.encoder.layer.11.attention.output.LayerNorm.bias requires_grad= True
deberta.encoder.layer.11.intermediate.dense.weight requires_grad= True
deberta.encoder.layer.11.intermediate.dense.bias requires_grad= True
deberta.encoder.layer.11.output.dense.weight requires_grad= True
deberta.encoder.layer.11.output.dense.bias requires_grad= True
deberta.encoder.layer.11.output.LayerNorm.weight requires_grad= True
deberta.encoder.layer.11.output.LayerNorm.bias requires_grad= True
deberta.encoder.rel_embeddings.weight requires_grad= True
deberta.encoder.LayerNorm.weight requires_grad= True
deberta.encoder.LayerNorm.bias requires_grad= True
pooler.dense.weight requires_grad= True
pooler.dense.bias requires_grad= True
classifier.weight requires_grad= True
classifier.bias requires_grad= True

```

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

```
[67]: # 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-20-0816cb2c5003>:31: FutureWarning: `tokenizer` is deprecated and
will be removed in version 5.0.0 for `Trainer.__init__`. Use `processing_class`
instead.
  trainer = Trainer(
model.safetensors:   0%|          | 0.00/371M [00:00<?, ?B/s]
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
Validation metrics: {'eval_loss': 0.7051265835762024, 'eval_accuracy':
0.48484848484848486, 'eval_precision': 0.5, 'eval_recall': 0.0196078431372549,
'eval_f1': 0.03773584905660377, 'eval_runtime': 5.9369,
'eval_samples_per_second': 16.675, 'eval_steps_per_second': 0.168, 'epoch': 1.0}
Test metrics: {'eval_loss': 0.7153680324554443, 'eval_accuracy':
0.44021739130434784, 'eval_precision': 0.25, 'eval_recall':
0.020202020202020204, 'eval_f1': 0.037383177570093455, 'eval_runtime': 5.4151,
'eval_samples_per_second': 33.979, 'eval_steps_per_second': 0.369, 'epoch': 1.0}

```

```

[68]: # 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\_microsoft/deberta-v3-base\_binary\_complexity\_2025-04-11T09:06:35.646933-07:00

<IPython.core.display.HTML object>

EXPERIMENT LOGGED TO:  
/content/drive/MyDrive/266-final/results/experiment\_runs.txt

**0.2.10** snc xlnet/xlnet-base-cased regularization\_weight\_decay = 0.5 learning\_rate = 5e-6 size\_batch = 128 length\_max = 128 num\_epochs = 1

```

[25]: # 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
# named_model = "answerdotai/ModernBERT-large" # modern bert
# named_model = "microsoft/deberta-v3-base" # deberta
named_model = "xlnet/xlnet-base-cased" #
# named_model = "xlnet/xlnet-large-cased" #
#####
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])

```



```

        1, 1, 1, 1, 1, 1, 1, 1]})
Loading from Hugging Face model: xlnet/xlnet-base-cased
config.json: 0%|          | 0.00/760 [00:00<?, ?B/s]
spiece.model: 0%|          | 0.00/798k [00:00<?, ?B/s]
tokenizer.json: 0%|          | 0.00/1.38M [00:00<?, ?B/s]
pytorch_model.bin: 0%|          | 0.00/467M [00:00<?, ?B/s]

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

```

```

=====
xlnet/xlnet-base-cased :
=====
num_parameters: 117310466
num_trainable_parameters at load: 117310466
=====
model lineage: {'type': 'huggingface_hub', 'path': 'xlnet/xlnet-base-cased',
'timestamp': '2025-04-11 16:07:56'}
=====

```

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

```

XLNetForSequenceClassification(
  (transformer): XLNetModel(
    (word_embedding): Embedding(32000, 768)
    (layer): ModuleList(
      (0-11): 12 x XLNetLayer(
        (rel_attn): XLNetRelativeAttention(
          (layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
        (ff): XLNetFeedForward(
          (layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
          (layer_1): Linear(in_features=768, out_features=3072, bias=True)
          (layer_2): Linear(in_features=3072, out_features=768, bias=True)
          (dropout): Dropout(p=0.1, inplace=False)
          (activation_function): GELUActivation()
        )
        (dropout): Dropout(p=0.1, inplace=False)
      )
    )
    (dropout): Dropout(p=0.1, inplace=False)
  )
)

```

```

(sequence_summary): SequenceSummary(
  (summary): Linear(in_features=768, out_features=768, bias=True)
  (activation): Tanh()
  (first_dropout): Identity()
  (last_dropout): Dropout(p=0.1, inplace=False)
)
(logits_proj): Linear(in_features=768, out_features=2, bias=True)
)

```

```

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

```

```

transformer.mask_emb requires_grad= True
transformer.word_embedding.weight requires_grad= True
transformer.layer.0.rel_attn.q requires_grad= True
transformer.layer.0.rel_attn.k requires_grad= True
transformer.layer.0.rel_attn.v requires_grad= True
transformer.layer.0.rel_attn.o requires_grad= True
transformer.layer.0.rel_attn.r requires_grad= True
transformer.layer.0.rel_attn.r_r_bias requires_grad= True
transformer.layer.0.rel_attn.r_s_bias requires_grad= True
transformer.layer.0.rel_attn.r_w_bias requires_grad= True
transformer.layer.0.rel_attn.seg_embed requires_grad= True
transformer.layer.0.rel_attn.layer_norm.weight requires_grad= True
transformer.layer.0.rel_attn.layer_norm.bias requires_grad= True
transformer.layer.0.ff.layer_norm.weight requires_grad= True
transformer.layer.0.ff.layer_norm.bias requires_grad= True
transformer.layer.0.ff.layer_1.weight requires_grad= True
transformer.layer.0.ff.layer_1.bias requires_grad= True
transformer.layer.0.ff.layer_2.weight requires_grad= True
transformer.layer.0.ff.layer_2.bias requires_grad= True
transformer.layer.1.rel_attn.q requires_grad= True
transformer.layer.1.rel_attn.k requires_grad= True
transformer.layer.1.rel_attn.v requires_grad= True
transformer.layer.1.rel_attn.o requires_grad= True
transformer.layer.1.rel_attn.r requires_grad= True
transformer.layer.1.rel_attn.r_r_bias requires_grad= True
transformer.layer.1.rel_attn.r_s_bias requires_grad= True
transformer.layer.1.rel_attn.r_w_bias requires_grad= True
transformer.layer.1.rel_attn.seg_embed requires_grad= True
transformer.layer.1.rel_attn.layer_norm.weight requires_grad= True
transformer.layer.1.rel_attn.layer_norm.bias requires_grad= True
transformer.layer.1.ff.layer_norm.weight requires_grad= True
transformer.layer.1.ff.layer_norm.bias requires_grad= True
transformer.layer.1.ff.layer_1.weight requires_grad= True
transformer.layer.1.ff.layer_1.bias requires_grad= True
transformer.layer.1.ff.layer_2.weight requires_grad= True
transformer.layer.1.ff.layer_2.bias requires_grad= True

```





transformer.layer.7.ff.layer\_norm.weight requires\_grad= True  
transformer.layer.7.ff.layer\_norm.bias requires\_grad= True  
transformer.layer.7.ff.layer\_1.weight requires\_grad= True  
transformer.layer.7.ff.layer\_1.bias requires\_grad= True  
transformer.layer.7.ff.layer\_2.weight requires\_grad= True  
transformer.layer.7.ff.layer\_2.bias requires\_grad= True  
transformer.layer.8.rel\_attn.q requires\_grad= True  
transformer.layer.8.rel\_attn.k requires\_grad= True  
transformer.layer.8.rel\_attn.v requires\_grad= True  
transformer.layer.8.rel\_attn.o requires\_grad= True  
transformer.layer.8.rel\_attn.r requires\_grad= True  
transformer.layer.8.rel\_attn.r\_r\_bias requires\_grad= True  
transformer.layer.8.rel\_attn.r\_s\_bias requires\_grad= True  
transformer.layer.8.rel\_attn.r\_w\_bias requires\_grad= True  
transformer.layer.8.rel\_attn.seg\_embed requires\_grad= True  
transformer.layer.8.rel\_attn.layer\_norm.weight requires\_grad= True  
transformer.layer.8.rel\_attn.layer\_norm.bias requires\_grad= True  
transformer.layer.8.ff.layer\_norm.weight requires\_grad= True  
transformer.layer.8.ff.layer\_norm.bias requires\_grad= True  
transformer.layer.8.ff.layer\_1.weight requires\_grad= True  
transformer.layer.8.ff.layer\_1.bias requires\_grad= True  
transformer.layer.8.ff.layer\_2.weight requires\_grad= True  
transformer.layer.8.ff.layer\_2.bias requires\_grad= True  
transformer.layer.9.rel\_attn.q requires\_grad= True  
transformer.layer.9.rel\_attn.k requires\_grad= True  
transformer.layer.9.rel\_attn.v requires\_grad= True  
transformer.layer.9.rel\_attn.o requires\_grad= True  
transformer.layer.9.rel\_attn.r requires\_grad= True  
transformer.layer.9.rel\_attn.r\_r\_bias requires\_grad= True  
transformer.layer.9.rel\_attn.r\_s\_bias requires\_grad= True  
transformer.layer.9.rel\_attn.r\_w\_bias requires\_grad= True  
transformer.layer.9.rel\_attn.seg\_embed requires\_grad= True  
transformer.layer.9.rel\_attn.layer\_norm.weight requires\_grad= True  
transformer.layer.9.rel\_attn.layer\_norm.bias requires\_grad= True  
transformer.layer.9.ff.layer\_norm.weight requires\_grad= True  
transformer.layer.9.ff.layer\_norm.bias requires\_grad= True  
transformer.layer.9.ff.layer\_1.weight requires\_grad= True  
transformer.layer.9.ff.layer\_1.bias requires\_grad= True  
transformer.layer.9.ff.layer\_2.weight requires\_grad= True  
transformer.layer.9.ff.layer\_2.bias requires\_grad= True  
transformer.layer.10.rel\_attn.q requires\_grad= True  
transformer.layer.10.rel\_attn.k requires\_grad= True  
transformer.layer.10.rel\_attn.v requires\_grad= True  
transformer.layer.10.rel\_attn.o requires\_grad= True  
transformer.layer.10.rel\_attn.r requires\_grad= True  
transformer.layer.10.rel\_attn.r\_r\_bias requires\_grad= True  
transformer.layer.10.rel\_attn.r\_s\_bias requires\_grad= True  
transformer.layer.10.rel\_attn.r\_w\_bias requires\_grad= True



```

transformer.layer.10.rel_attn(seg_embed requires_grad= True
transformer.layer.10.rel_attn.layer_norm.weight requires_grad= True
transformer.layer.10.rel_attn.layer_norm.bias requires_grad= True
transformer.layer.10.ff.layer_norm.weight requires_grad= True
transformer.layer.10.ff.layer_norm.bias requires_grad= True
transformer.layer.10.ff.layer_1.weight requires_grad= True
transformer.layer.10.ff.layer_1.bias requires_grad= True
transformer.layer.10.ff.layer_2.weight requires_grad= True
transformer.layer.10.ff.layer_2.bias requires_grad= True
transformer.layer.11.rel_attn.q requires_grad= True
transformer.layer.11.rel_attn.k requires_grad= True
transformer.layer.11.rel_attn.v requires_grad= True
transformer.layer.11.rel_attn.o requires_grad= True
transformer.layer.11.rel_attn.r requires_grad= True
transformer.layer.11.rel_attn.r_r_bias requires_grad= True
transformer.layer.11.rel_attn.r_s_bias requires_grad= True
transformer.layer.11.rel_attn.r_w_bias requires_grad= True
transformer.layer.11.rel_attn(seg_embed requires_grad= True
transformer.layer.11.rel_attn.layer_norm.weight requires_grad= True
transformer.layer.11.rel_attn.layer_norm.bias requires_grad= True
transformer.layer.11.ff.layer_norm.weight requires_grad= True
transformer.layer.11.ff.layer_norm.bias requires_grad= True
transformer.layer.11.ff.layer_1.weight requires_grad= True
transformer.layer.11.ff.layer_1.bias requires_grad= True
transformer.layer.11.ff.layer_2.weight requires_grad= True
transformer.layer.11.ff.layer_2.bias requires_grad= True
sequence_summary.summary.weight requires_grad= True
sequence_summary.summary.bias requires_grad= True
logits_proj.weight requires_grad= True
logits_proj.bias requires_grad= True

```

```

[28]: #####
layers_to_unfreeze = [
    "transformer.layer.11.rel_attn.q",
    "transformer.layer.11.rel_attn.k",
    "transformer.layer.11.rel_attn.v",
    "transformer.layer.11.rel_attn.o",
    "transformer.layer.11.rel_attn.r",
    "transformer.layer.11.rel_attn.r_r_bias",
    "transformer.layer.11.rel_attn.r_s_bias",
    "transformer.layer.11.rel_attn.r_w_bias",
    "transformer.layer.11.rel_attn(seg_embed",
    "transformer.layer.11.rel_attn.layer_norm.weight",
    "transformer.layer.11.rel_attn.layer_norm.bias",
    "transformer.layer.11.ff.layer_norm.weight",
    "transformer.layer.11.ff.layer_norm.bias",
    "transformer.layer.11.ff.layer_1.weight",

```

```

"transformer.layer.11.ff.layer_1.bias",
"transformer.layer.11.ff.layer_2.weight",
"transformer.layer.11.ff.layer_2.bias",
"sequence_summary.summary.weight",
"sequence_summary.summary.bias",
"logits_proj.weight",
"logits_proj.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))

```

```

XLNetConfig {
  "_attn_implementation_autoset": true,
  "architectures": [
    "XLNetLMHeadModel"
  ],
  "attn_type": "bi",
  "bi_data": false,
  "bos_token_id": 1,
  "clamp_len": -1,
  "d_head": 64,
  "d_inner": 3072,
  "d_model": 768,
  "dropout": 0.1,
  "end_n_top": 5,
  "eos_token_id": 2,
  "ff_activation": "gelu",
  "initializer_range": 0.02,
  "layer_norm_eps": 1e-12,
  "mem_len": null,

```

```

"model_type": "xlnet",
"n_head": 12,
"n_layer": 12,
"pad_token_id": 5,
"reuse_len": null,
"same_length": false,
"start_n_top": 5,
"summary_activation": "tanh",
"summary_last_dropout": 0.1,
"summary_type": "last",
"summary_use_proj": true,
"task_specific_params": {
    "text-generation": {
        "do_sample": true,
        "max_length": 250
    }
},
"torch_dtype": "float32",
"transformers_version": "4.50.3",
"untie_r": true,
"use_mems_eval": true,
"use_mems_train": false,
"vocab_size": 32000
}

```

=====

```

num_parameters: 117310466
num_trainable_parameters: 8270594

```

=====

Experiment configuration used with this experiment:

```

model used: xlnet/xlnet-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: snc_morph_seq

```

=====

```

num_trainable_parameters: 8270594

```

```

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

```

```

transformer.mask_emb requires_grad= False
transformer.word_embedding.weight requires_grad= False
transformer.layer.0.rel_attn.q requires_grad= False

```







```

transformer.layer.8.rel_attn.layer_norm.weight requires_grad= False
transformer.layer.8.rel_attn.layer_norm.bias requires_grad= False
transformer.layer.8.ff.layer_norm.weight requires_grad= False
transformer.layer.8.ff.layer_norm.bias requires_grad= False
transformer.layer.8.ff.layer_1.weight requires_grad= False
transformer.layer.8.ff.layer_1.bias requires_grad= False
transformer.layer.8.ff.layer_2.weight requires_grad= False
transformer.layer.8.ff.layer_2.bias requires_grad= False
transformer.layer.9.rel_attn.q requires_grad= False
transformer.layer.9.rel_attn.k requires_grad= False
transformer.layer.9.rel_attn.v requires_grad= False
transformer.layer.9.rel_attn.o requires_grad= False
transformer.layer.9.rel_attn.r requires_grad= False
transformer.layer.9.rel_attn.r_r_bias requires_grad= False
transformer.layer.9.rel_attn.r_s_bias requires_grad= False
transformer.layer.9.rel_attn.r_w_bias requires_grad= False
transformer.layer.9.rel_attn.seg_embed requires_grad= False
transformer.layer.9.rel_attn.layer_norm.weight requires_grad= False
transformer.layer.9.rel_attn.layer_norm.bias requires_grad= False
transformer.layer.9.ff.layer_norm.weight requires_grad= False
transformer.layer.9.ff.layer_norm.bias requires_grad= False
transformer.layer.9.ff.layer_1.weight requires_grad= False
transformer.layer.9.ff.layer_1.bias requires_grad= False
transformer.layer.9.ff.layer_2.weight requires_grad= False
transformer.layer.9.ff.layer_2.bias requires_grad= False
transformer.layer.10.rel_attn.q requires_grad= False
transformer.layer.10.rel_attn.k requires_grad= False
transformer.layer.10.rel_attn.v requires_grad= False
transformer.layer.10.rel_attn.o requires_grad= False
transformer.layer.10.rel_attn.r requires_grad= False
transformer.layer.10.rel_attn.r_r_bias requires_grad= False
transformer.layer.10.rel_attn.r_s_bias requires_grad= False
transformer.layer.10.rel_attn.r_w_bias requires_grad= False
transformer.layer.10.rel_attn.seg_embed requires_grad= False
transformer.layer.10.rel_attn.layer_norm.weight requires_grad= False
transformer.layer.10.rel_attn.layer_norm.bias requires_grad= False
transformer.layer.10.ff.layer_norm.weight requires_grad= False
transformer.layer.10.ff.layer_norm.bias requires_grad= False
transformer.layer.10.ff.layer_1.weight requires_grad= False
transformer.layer.10.ff.layer_1.bias requires_grad= False
transformer.layer.10.ff.layer_2.weight requires_grad= False
transformer.layer.10.ff.layer_2.bias requires_grad= False
transformer.layer.11.rel_attn.q requires_grad= True
transformer.layer.11.rel_attn.k requires_grad= True
transformer.layer.11.rel_attn.v requires_grad= True
transformer.layer.11.rel_attn.o requires_grad= True
transformer.layer.11.rel_attn.r requires_grad= True
transformer.layer.11.rel_attn.r_r_bias requires_grad= True

```

```

transformer.layer.11.rel_attn.r_s_bias requires_grad= True
transformer.layer.11.rel_attn.r_w_bias requires_grad= True
transformer.layer.11.rel_attn.seg_embed requires_grad= True
transformer.layer.11.rel_attn.layer_norm.weight requires_grad= True
transformer.layer.11.rel_attn.layer_norm.bias requires_grad= True
transformer.layer.11.ff.layer_norm.weight requires_grad= True
transformer.layer.11.ff.layer_norm.bias requires_grad= True
transformer.layer.11.ff.layer_1.weight requires_grad= True
transformer.layer.11.ff.layer_1.bias requires_grad= True
transformer.layer.11.ff.layer_2.weight requires_grad= True
transformer.layer.11.ff.layer_2.bias requires_grad= True
sequence_summary.summary.weight requires_grad= True
sequence_summary.summary.bias requires_grad= True
logits_proj.weight requires_grad= True
logits_proj.bias requires_grad= True

```

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

```
[31]: # 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-20-0816cb2c5003>: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>
model.safetensors:   0%|          | 0.00/467M [00:00<?, ?B/s]
<IPython.core.display.HTML object>
Validation metrics: {'eval_loss': 0.6875008344650269, 'eval_accuracy':

```



```
0.5353535353535354, 'eval_precision': 0.5352112676056338, 'eval_recall':
0.7450980392156863, 'eval_f1': 0.6229508196721312, 'eval_runtime': 4.9977,
'eval_samples_per_second': 19.809, 'eval_steps_per_second': 0.2, 'epoch': 1.0}
Test metrics: {'eval_loss': 0.6866654753684998, 'eval_accuracy':
0.5380434782608695, 'eval_precision': 0.5583333333333333, 'eval_recall':
0.6767676767676768, 'eval_f1': 0.6118721461187214, 'eval_runtime': 5.4763,
'eval_samples_per_second': 33.6, 'eval_steps_per_second': 0.365, 'epoch': 1.0}
```

```
[32]: # 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\_xlnet/xlnet-base-  
cased\_binary\_complexity\_2025-04-11T09:08:16.958263-07:00

<IPython.core.display.HTML object>

EXPERIMENT LOGGED TO:  
/content/drive/MyDrive/266-final/results/experiment\_runs.txt

0.2.11 snc xlnet/xlnet-large-cased regularization\_weight\_decay = 0.5 learning\_rate = 5e-6 size\_batch = 128 length\_max = 128 num\_epochs = 1

```
[33]: # 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
# named_model = "answerdotai/ModernBERT-large" # modern bert
# named_model = "microsoft/deberta-v3-base" # deberta
# named_model = "xlnet/xlnet-base-cased" #
named_model = "xlnet/xlnet-large-cased" #
#####
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="xlnet/xlnet-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%|          | 0/1517 [00:00<?, ? examples/s]
Map:   0%|          | 0/99 [00:00<?, ? examples/s]
Map:   0%|          | 0/184 [00:00<?, ? examples/s]

```

Datasets prepared. Sample from train\_data\_hf:

[illegible]

Loading from Hugging Face model: xlnet/xlnet-large-cased

```
config.json: 0%|          | 0.00/761 [00:00<?, ?B/s]
```

```
spiece.model: 0%|          | 0.00/798k [00:00<?, ?B/s]
```

```
tokenizer.json: 0%|          | 0.00/1.38M [00:00<?, ?B/s]
```

```
pytorch_model.bin: 0%|          | 0.00/1.44G [00:00<?, ?B/s]
```

Some weights of `XLNetForSequenceClassification` were not initialized from the model checkpoint at `xlnet/xlnet-large-cased` and are newly initialized:

```
['logits_proj.bias', 'logits_proj.weight', 'sequence_summary.summary.bias',
'sequence_summary.summary.weight']
```

You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.

=====

xlnet/xlnet-large-cased :

=====

```
num_parameters: 361320450
```

```
num_trainable_parameters at load: 361320450
```

=====

```
model lineage: {'type': 'huggingface_hub', 'path': 'xlnet/xlnet-large-cased',
'timestamp': '2025-04-11 16:08:52'}
```

=====

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

```

XLNetForSequenceClassification(
  (transformer): XLNetModel(
    (word_embedding): Embedding(32000, 1024)
    (layer): ModuleList(
      (0-23): 24 x XLNetLayer(
        (rel_attn): XLNetRelativeAttention(
          (layer_norm): LayerNorm((1024,)), eps=1e-12, elementwise_affine=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
        (ff): XLNetFeedForward(
          (layer_norm): LayerNorm((1024,)), eps=1e-12, elementwise_affine=True)
          (layer_1): Linear(in_features=1024, out_features=4096, bias=True)
          (layer_2): Linear(in_features=4096, out_features=1024, bias=True)
          (dropout): Dropout(p=0.1, inplace=False)
          (activation_function): GELUActivation()
        )
        (dropout): Dropout(p=0.1, inplace=False)
      )
    )
    (dropout): Dropout(p=0.1, inplace=False)
  )
  (sequence_summary): SequenceSummary(
    (summary): Linear(in_features=1024, out_features=1024, bias=True)
    (activation): Tanh()
    (first_dropout): Identity()
    (last_dropout): Dropout(p=0.1, inplace=False)
  )
  (logits_proj): Linear(in_features=1024, out_features=2, bias=True)
)

```

```

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

```

```

transformer.mask_emb requires_grad= True
transformer.word_embedding.weight requires_grad= True
transformer.layer.0.rel_attn.q requires_grad= True
transformer.layer.0.rel_attn.k requires_grad= True
transformer.layer.0.rel_attn.v requires_grad= True
transformer.layer.0.rel_attn.o requires_grad= True
transformer.layer.0.rel_attn.r requires_grad= True
transformer.layer.0.rel_attn.r_r_bias requires_grad= True
transformer.layer.0.rel_attn.r_s_bias requires_grad= True
transformer.layer.0.rel_attn.r_w_bias requires_grad= True
transformer.layer.0.rel_attn.seg_embed requires_grad= True
transformer.layer.0.rel_attn.layer_norm.weight requires_grad= True
transformer.layer.0.rel_attn.layer_norm.bias requires_grad= True
transformer.layer.0.ff.layer_norm.weight requires_grad= True
transformer.layer.0.ff.layer_norm.bias requires_grad= True

```



```
transformer.layer.3.rel_attn.layer_norm.bias requires_grad= True
transformer.layer.3.ff.layer_norm.weight requires_grad= True
transformer.layer.3.ff.layer_norm.bias requires_grad= True
transformer.layer.3.ff.layer_1.weight requires_grad= True
transformer.layer.3.ff.layer_1.bias requires_grad= True
transformer.layer.3.ff.layer_2.weight requires_grad= True
transformer.layer.3.ff.layer_2.bias requires_grad= True
transformer.layer.4.rel_attn.q requires_grad= True
transformer.layer.4.rel_attn.k requires_grad= True
transformer.layer.4.rel_attn.v requires_grad= True
transformer.layer.4.rel_attn.o requires_grad= True
transformer.layer.4.rel_attn.r requires_grad= True
transformer.layer.4.rel_attn.r_r_bias requires_grad= True
transformer.layer.4.rel_attn.r_s_bias requires_grad= True
transformer.layer.4.rel_attn.r_w_bias requires_grad= True
transformer.layer.4.rel_attn.seg_embed requires_grad= True
transformer.layer.4.rel_attn.layer_norm.weight requires_grad= True
transformer.layer.4.rel_attn.layer_norm.bias requires_grad= True
transformer.layer.4.ff.layer_norm.weight requires_grad= True
transformer.layer.4.ff.layer_norm.bias requires_grad= True
transformer.layer.4.ff.layer_1.weight requires_grad= True
transformer.layer.4.ff.layer_1.bias requires_grad= True
transformer.layer.4.ff.layer_2.weight requires_grad= True
transformer.layer.4.ff.layer_2.bias requires_grad= True
transformer.layer.5.rel_attn.q requires_grad= True
transformer.layer.5.rel_attn.k requires_grad= True
transformer.layer.5.rel_attn.v requires_grad= True
transformer.layer.5.rel_attn.o requires_grad= True
transformer.layer.5.rel_attn.r requires_grad= True
transformer.layer.5.rel_attn.r_r_bias requires_grad= True
transformer.layer.5.rel_attn.r_s_bias requires_grad= True
transformer.layer.5.rel_attn.r_w_bias requires_grad= True
transformer.layer.5.rel_attn.seg_embed requires_grad= True
transformer.layer.5.rel_attn.layer_norm.weight requires_grad= True
transformer.layer.5.rel_attn.layer_norm.bias requires_grad= True
transformer.layer.5.ff.layer_norm.weight requires_grad= True
transformer.layer.5.ff.layer_norm.bias requires_grad= True
transformer.layer.5.ff.layer_1.weight requires_grad= True
transformer.layer.5.ff.layer_1.bias requires_grad= True
transformer.layer.5.ff.layer_2.weight requires_grad= True
transformer.layer.5.ff.layer_2.bias requires_grad= True
transformer.layer.6.rel_attn.q requires_grad= True
transformer.layer.6.rel_attn.k requires_grad= True
transformer.layer.6.rel_attn.v requires_grad= True
transformer.layer.6.rel_attn.o requires_grad= True
transformer.layer.6.rel_attn.r requires_grad= True
transformer.layer.6.rel_attn.r_r_bias requires_grad= True
transformer.layer.6.rel_attn.r s bias requires grad= True
```





transformer.layer.9.rel\_attn.r requires\_grad= True  
transformer.layer.9.rel\_attn.r\_r\_bias requires\_grad= True  
transformer.layer.9.rel\_attn.r\_s\_bias requires\_grad= True  
transformer.layer.9.rel\_attn.r\_w\_bias requires\_grad= True  
transformer.layer.9.rel\_attn.seg\_embed requires\_grad= True  
transformer.layer.9.rel\_attn.layer\_norm.weight requires\_grad= True  
transformer.layer.9.rel\_attn.layer\_norm.bias requires\_grad= True  
transformer.layer.9.ff.layer\_norm.weight requires\_grad= True  
transformer.layer.9.ff.layer\_norm.bias requires\_grad= True  
transformer.layer.9.ff.layer\_1.weight requires\_grad= True  
transformer.layer.9.ff.layer\_1.bias requires\_grad= True  
transformer.layer.9.ff.layer\_2.weight requires\_grad= True  
transformer.layer.9.ff.layer\_2.bias requires\_grad= True  
transformer.layer.10.rel\_attn.q requires\_grad= True  
transformer.layer.10.rel\_attn.k requires\_grad= True  
transformer.layer.10.rel\_attn.v requires\_grad= True  
transformer.layer.10.rel\_attn.o requires\_grad= True  
transformer.layer.10.rel\_attn.r requires\_grad= True  
transformer.layer.10.rel\_attn.r\_r\_bias requires\_grad= True  
transformer.layer.10.rel\_attn.r\_s\_bias requires\_grad= True  
transformer.layer.10.rel\_attn.r\_w\_bias requires\_grad= True  
transformer.layer.10.rel\_attn.seg\_embed requires\_grad= True  
transformer.layer.10.rel\_attn.layer\_norm.weight requires\_grad= True  
transformer.layer.10.rel\_attn.layer\_norm.bias requires\_grad= True  
transformer.layer.10.ff.layer\_norm.weight requires\_grad= True  
transformer.layer.10.ff.layer\_norm.bias requires\_grad= True  
transformer.layer.10.ff.layer\_1.weight requires\_grad= True  
transformer.layer.10.ff.layer\_1.bias requires\_grad= True  
transformer.layer.10.ff.layer\_2.weight requires\_grad= True  
transformer.layer.10.ff.layer\_2.bias requires\_grad= True  
transformer.layer.11.rel\_attn.q requires\_grad= True  
transformer.layer.11.rel\_attn.k requires\_grad= True  
transformer.layer.11.rel\_attn.v requires\_grad= True  
transformer.layer.11.rel\_attn.o requires\_grad= True  
transformer.layer.11.rel\_attn.r requires\_grad= True  
transformer.layer.11.rel\_attn.r\_r\_bias requires\_grad= True  
transformer.layer.11.rel\_attn.r\_s\_bias requires\_grad= True  
transformer.layer.11.rel\_attn.r\_w\_bias requires\_grad= True  
transformer.layer.11.rel\_attn.seg\_embed requires\_grad= True  
transformer.layer.11.rel\_attn.layer\_norm.weight requires\_grad= True  
transformer.layer.11.rel\_attn.layer\_norm.bias requires\_grad= True  
transformer.layer.11.ff.layer\_norm.weight requires\_grad= True  
transformer.layer.11.ff.layer\_norm.bias requires\_grad= True  
transformer.layer.11.ff.layer\_1.weight requires\_grad= True  
transformer.layer.11.ff.layer\_1.bias requires\_grad= True  
transformer.layer.11.ff.layer\_2.weight requires\_grad= True  
transformer.layer.11.ff.layer\_2.bias requires\_grad= True  
transformer.layer.12.rel\_attn.q requires\_grad= True





transformer.layer.17.ff.layer\_norm.bias requires\_grad= True  
transformer.layer.17.ff.layer\_1.weight requires\_grad= True  
transformer.layer.17.ff.layer\_1.bias requires\_grad= True  
transformer.layer.17.ff.layer\_2.weight requires\_grad= True  
transformer.layer.17.ff.layer\_2.bias requires\_grad= True  
transformer.layer.18.rel\_attn.q requires\_grad= True  
transformer.layer.18.rel\_attn.k requires\_grad= True  
transformer.layer.18.rel\_attn.v requires\_grad= True  
transformer.layer.18.rel\_attn.o requires\_grad= True  
transformer.layer.18.rel\_attn.r requires\_grad= True  
transformer.layer.18.rel\_attn.r\_r\_bias requires\_grad= True  
transformer.layer.18.rel\_attn.r\_s\_bias requires\_grad= True  
transformer.layer.18.rel\_attn.r\_w\_bias requires\_grad= True  
transformer.layer.18.rel\_attn.seg\_embed requires\_grad= True  
transformer.layer.18.rel\_attn.layer\_norm.weight requires\_grad= True  
transformer.layer.18.rel\_attn.layer\_norm.bias requires\_grad= True  
transformer.layer.18.ff.layer\_norm.weight requires\_grad= True  
transformer.layer.18.ff.layer\_norm.bias requires\_grad= True  
transformer.layer.18.ff.layer\_1.weight requires\_grad= True  
transformer.layer.18.ff.layer\_1.bias requires\_grad= True  
transformer.layer.18.ff.layer\_2.weight requires\_grad= True  
transformer.layer.18.ff.layer\_2.bias requires\_grad= True  
transformer.layer.19.rel\_attn.q requires\_grad= True  
transformer.layer.19.rel\_attn.k requires\_grad= True  
transformer.layer.19.rel\_attn.v requires\_grad= True  
transformer.layer.19.rel\_attn.o requires\_grad= True  
transformer.layer.19.rel\_attn.r requires\_grad= True  
transformer.layer.19.rel\_attn.r\_r\_bias requires\_grad= True  
transformer.layer.19.rel\_attn.r\_s\_bias requires\_grad= True  
transformer.layer.19.rel\_attn.r\_w\_bias requires\_grad= True  
transformer.layer.19.rel\_attn.seg\_embed requires\_grad= True  
transformer.layer.19.rel\_attn.layer\_norm.weight requires\_grad= True  
transformer.layer.19.rel\_attn.layer\_norm.bias requires\_grad= True  
transformer.layer.19.ff.layer\_norm.weight requires\_grad= True  
transformer.layer.19.ff.layer\_norm.bias requires\_grad= True  
transformer.layer.19.ff.layer\_1.weight requires\_grad= True  
transformer.layer.19.ff.layer\_1.bias requires\_grad= True  
transformer.layer.19.ff.layer\_2.weight requires\_grad= True  
transformer.layer.19.ff.layer\_2.bias requires\_grad= True  
transformer.layer.20.rel\_attn.q requires\_grad= True  
transformer.layer.20.rel\_attn.k requires\_grad= True  
transformer.layer.20.rel\_attn.v requires\_grad= True  
transformer.layer.20.rel\_attn.o requires\_grad= True  
transformer.layer.20.rel\_attn.r requires\_grad= True  
transformer.layer.20.rel\_attn.r\_r\_bias requires\_grad= True  
transformer.layer.20.rel\_attn.r\_s\_bias requires\_grad= True  
transformer.layer.20.rel\_attn.r\_w\_bias requires\_grad= True  
transformer.layer.20.rel\_attn.seg\_embed requires\_grad= True

[illegible]

```

transformer.layer.23.rel_attn.r_s_bias requires_grad= True
transformer.layer.23.rel_attn.r_w_bias requires_grad= True
transformer.layer.23.rel_attn.seg_embed requires_grad= True
transformer.layer.23.rel_attn.layer_norm.weight requires_grad= True
transformer.layer.23.rel_attn.layer_norm.bias requires_grad= True
transformer.layer.23.ff.layer_norm.weight requires_grad= True
transformer.layer.23.ff.layer_norm.bias requires_grad= True
transformer.layer.23.ff.layer_1.weight requires_grad= True
transformer.layer.23.ff.layer_1.bias requires_grad= True
transformer.layer.23.ff.layer_2.weight requires_grad= True
transformer.layer.23.ff.layer_2.bias requires_grad= True
sequence_summary.summary.weight requires_grad= True
sequence_summary.summary.bias requires_grad= True
logits_proj.weight requires_grad= True
logits_proj.bias requires_grad= True

```

```

[36]: #####
layers_to_unfreeze = [
    "transformer.layer.23.rel_attn.q",
    "transformer.layer.23.rel_attn.k",
    "transformer.layer.23.rel_attn.v",
    "transformer.layer.23.rel_attn.o",
    "transformer.layer.23.rel_attn.r",
    "transformer.layer.23.rel_attn.r_r_bias",
    "transformer.layer.23.rel_attn.r_s_bias",
    "transformer.layer.23.rel_attn.r_w_bias",
    "transformer.layer.23.rel_attn.seg_embed",
    "transformer.layer.23.rel_attn.layer_norm.weight",
    "transformer.layer.23.rel_attn.layer_norm.bias",
    "transformer.layer.23.ff.layer_norm.weight",
    "transformer.layer.23.ff.layer_norm.bias",
    "transformer.layer.23.ff.layer_1.weight",
    "transformer.layer.23.ff.layer_1.bias",
    "transformer.layer.23.ff.layer_2.weight",
    "transformer.layer.23.ff.layer_2.bias",
    "sequence_summary.summary.weight",
    "sequence_summary.summary.bias",
    "logits_proj.weight",
    "logits_proj.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))

```

```

XLNetConfig {
  "_attn_implementation_autoset": true,
  "architectures": [
    "XLNetLMHeadModel"
  ],
  "attn_type": "bi",
  "bi_data": false,
  "bos_token_id": 1,
  "clamp_len": -1,
  "d_head": 64,
  "d_inner": 4096,
  "d_model": 1024,
  "dropout": 0.1,
  "end_n_top": 5,
  "eos_token_id": 2,
  "ff_activation": "gelu",
  "initializer_range": 0.02,
  "layer_norm_eps": 1e-12,
  "mem_len": null,
  "model_type": "xlnet",
  "n_head": 16,
  "n_layer": 24,
  "pad_token_id": 5,
  "reuse_len": null,
  "same_length": false,
  "start_n_top": 5,
  "summary_activation": "tanh",
  "summary_last_dropout": 0.1,
  "summary_type": "last",
  "summary_use_proj": true,
  "task_specific_params": {
    "text-generation": {
      "do_sample": true,
      "max_length": 250
    }
  }
}

```

```

    }
},
"torch_dtype": "float32",
"transformers_version": "4.50.3",
"untie_r": true,
"use_mems_eval": true,
"use_mems_train": false,
"vocab_size": 32000
}

```

```

=====

```

```

num_parameters: 361320450
num_trainable_parameters: 14697474

```

```

=====

```

Experiment configuration used with this experiment:

model used: xlnet/xlnet-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: snc\_morph\_seq

```

=====

```

```

num_trainable_parameters: 14697474

```

```

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

```

```

transformer.mask_emb requires_grad= False
transformer.word_embedding.weight requires_grad= False
transformer.layer.0.rel_attn.q requires_grad= False
transformer.layer.0.rel_attn.k requires_grad= False
transformer.layer.0.rel_attn.v requires_grad= False
transformer.layer.0.rel_attn.o requires_grad= False
transformer.layer.0.rel_attn.r requires_grad= False
transformer.layer.0.rel_attn.r_r_bias requires_grad= False
transformer.layer.0.rel_attn.r_s_bias requires_grad= False
transformer.layer.0.rel_attn.r_w_bias requires_grad= False
transformer.layer.0.rel_attn.seg_embed requires_grad= False
transformer.layer.0.rel_attn.layer_norm.weight requires_grad= False
transformer.layer.0.rel_attn.layer_norm.bias requires_grad= False
transformer.layer.0.ff.layer_norm.weight requires_grad= False
transformer.layer.0.ff.layer_norm.bias requires_grad= False
transformer.layer.0.ff.layer_1.weight requires_grad= False
transformer.layer.0.ff.layer_1.bias requires_grad= False
transformer.layer.0.ff.layer_2.weight requires_grad= False

```





[illegible]

[illegible]

```

transformer.layer.9.rel_attn.r_w_bias requires_grad= False
transformer.layer.9.rel_attn.seg_embed requires_grad= False
transformer.layer.9.rel_attn.layer_norm.weight requires_grad= False
transformer.layer.9.rel_attn.layer_norm.bias requires_grad= False
transformer.layer.9.ff.layer_norm.weight requires_grad= False
transformer.layer.9.ff.layer_norm.bias requires_grad= False
transformer.layer.9.ff.layer_1.weight requires_grad= False
transformer.layer.9.ff.layer_1.bias requires_grad= False
transformer.layer.9.ff.layer_2.weight requires_grad= False
transformer.layer.9.ff.layer_2.bias requires_grad= False
transformer.layer.10.rel_attn.q requires_grad= False
transformer.layer.10.rel_attn.k requires_grad= False
transformer.layer.10.rel_attn.v requires_grad= False
transformer.layer.10.rel_attn.o requires_grad= False
transformer.layer.10.rel_attn.r requires_grad= False
transformer.layer.10.rel_attn.r_r_bias requires_grad= False
transformer.layer.10.rel_attn.r_s_bias requires_grad= False
transformer.layer.10.rel_attn.r_w_bias requires_grad= False
transformer.layer.10.rel_attn.seg_embed requires_grad= False
transformer.layer.10.rel_attn.layer_norm.weight requires_grad= False
transformer.layer.10.rel_attn.layer_norm.bias requires_grad= False
transformer.layer.10.ff.layer_norm.weight requires_grad= False
transformer.layer.10.ff.layer_norm.bias requires_grad= False
transformer.layer.10.ff.layer_1.weight requires_grad= False
transformer.layer.10.ff.layer_1.bias requires_grad= False
transformer.layer.10.ff.layer_2.weight requires_grad= False
transformer.layer.10.ff.layer_2.bias requires_grad= False
transformer.layer.11.rel_attn.q requires_grad= False
transformer.layer.11.rel_attn.k requires_grad= False
transformer.layer.11.rel_attn.v requires_grad= False
transformer.layer.11.rel_attn.o requires_grad= False
transformer.layer.11.rel_attn.r requires_grad= False
transformer.layer.11.rel_attn.r_r_bias requires_grad= False
transformer.layer.11.rel_attn.r_s_bias requires_grad= False
transformer.layer.11.rel_attn.r_w_bias requires_grad= False
transformer.layer.11.rel_attn.seg_embed requires_grad= False
transformer.layer.11.rel_attn.layer_norm.weight requires_grad= False
transformer.layer.11.rel_attn.layer_norm.bias requires_grad= False
transformer.layer.11.ff.layer_norm.weight requires_grad= False
transformer.layer.11.ff.layer_norm.bias requires_grad= False
transformer.layer.11.ff.layer_1.weight requires_grad= False
transformer.layer.11.ff.layer_1.bias requires_grad= False
transformer.layer.11.ff.layer_2.weight requires_grad= False
transformer.layer.11.ff.layer_2.bias requires_grad= False
transformer.layer.12.rel_attn.q requires_grad= False
transformer.layer.12.rel_attn.k requires_grad= False
transformer.layer.12.rel_attn.v requires_grad= False
transformer.layer.12.rel_attn.o requires_grad= False

```

[illegible]



transformer.layer.17.ff.layer\_2.weight requires\_grad= False  
transformer.layer.17.ff.layer\_2.bias requires\_grad= False  
transformer.layer.18.rel\_attn.q requires\_grad= False  
transformer.layer.18.rel\_attn.k requires\_grad= False  
transformer.layer.18.rel\_attn.v requires\_grad= False  
transformer.layer.18.rel\_attn.o requires\_grad= False  
transformer.layer.18.rel\_attn.r requires\_grad= False  
transformer.layer.18.rel\_attn.r\_r\_bias requires\_grad= False  
transformer.layer.18.rel\_attn.r\_s\_bias requires\_grad= False  
transformer.layer.18.rel\_attn.r\_w\_bias requires\_grad= False  
transformer.layer.18.rel\_attn.seg\_embed requires\_grad= False  
transformer.layer.18.rel\_attn.layer\_norm.weight requires\_grad= False  
transformer.layer.18.rel\_attn.layer\_norm.bias requires\_grad= False  
transformer.layer.18.ff.layer\_norm.weight requires\_grad= False  
transformer.layer.18.ff.layer\_norm.bias requires\_grad= False  
transformer.layer.18.ff.layer\_1.weight requires\_grad= False  
transformer.layer.18.ff.layer\_1.bias requires\_grad= False  
transformer.layer.18.ff.layer\_2.weight requires\_grad= False  
transformer.layer.18.ff.layer\_2.bias requires\_grad= False  
transformer.layer.19.rel\_attn.q requires\_grad= False  
transformer.layer.19.rel\_attn.k requires\_grad= False  
transformer.layer.19.rel\_attn.v requires\_grad= False  
transformer.layer.19.rel\_attn.o requires\_grad= False  
transformer.layer.19.rel\_attn.r requires\_grad= False  
transformer.layer.19.rel\_attn.r\_r\_bias requires\_grad= False  
transformer.layer.19.rel\_attn.r\_s\_bias requires\_grad= False  
transformer.layer.19.rel\_attn.r\_w\_bias requires\_grad= False  
transformer.layer.19.rel\_attn.seg\_embed requires\_grad= False  
transformer.layer.19.rel\_attn.layer\_norm.weight requires\_grad= False  
transformer.layer.19.rel\_attn.layer\_norm.bias requires\_grad= False  
transformer.layer.19.ff.layer\_norm.weight requires\_grad= False  
transformer.layer.19.ff.layer\_norm.bias requires\_grad= False  
transformer.layer.19.ff.layer\_1.weight requires\_grad= False  
transformer.layer.19.ff.layer\_1.bias requires\_grad= False  
transformer.layer.19.ff.layer\_2.weight requires\_grad= False  
transformer.layer.19.ff.layer\_2.bias requires\_grad= False  
transformer.layer.20.rel\_attn.q requires\_grad= False  
transformer.layer.20.rel\_attn.k requires\_grad= False  
transformer.layer.20.rel\_attn.v requires\_grad= False  
transformer.layer.20.rel\_attn.o requires\_grad= False  
transformer.layer.20.rel\_attn.r requires\_grad= False  
transformer.layer.20.rel\_attn.r\_r\_bias requires\_grad= False  
transformer.layer.20.rel\_attn.r\_s\_bias requires\_grad= False  
transformer.layer.20.rel\_attn.r\_w\_bias requires\_grad= False  
transformer.layer.20.rel\_attn.seg\_embed requires\_grad= False  
transformer.layer.20.rel\_attn.layer\_norm.weight requires\_grad= False  
transformer.layer.20.rel\_attn.layer\_norm.bias requires\_grad= False  
transformer.layer.20.ff.layer\_norm.weight requires\_grad= False

```

transformer.layer.20.ff.layer_norm.bias requires_grad= False
transformer.layer.20.ff.layer_1.weight requires_grad= False
transformer.layer.20.ff.layer_1.bias requires_grad= False
transformer.layer.20.ff.layer_2.weight requires_grad= False
transformer.layer.20.ff.layer_2.bias requires_grad= False
transformer.layer.21.rel_attn.q requires_grad= False
transformer.layer.21.rel_attn.k requires_grad= False
transformer.layer.21.rel_attn.v requires_grad= False
transformer.layer.21.rel_attn.o requires_grad= False
transformer.layer.21.rel_attn.r requires_grad= False
transformer.layer.21.rel_attn.r_r_bias requires_grad= False
transformer.layer.21.rel_attn.r_s_bias requires_grad= False
transformer.layer.21.rel_attn.r_w_bias requires_grad= False
transformer.layer.21.rel_attn.seg_embed requires_grad= False
transformer.layer.21.rel_attn.layer_norm.weight requires_grad= False
transformer.layer.21.rel_attn.layer_norm.bias requires_grad= False
transformer.layer.21.ff.layer_norm.weight requires_grad= False
transformer.layer.21.ff.layer_norm.bias requires_grad= False
transformer.layer.21.ff.layer_1.weight requires_grad= False
transformer.layer.21.ff.layer_1.bias requires_grad= False
transformer.layer.21.ff.layer_2.weight requires_grad= False
transformer.layer.21.ff.layer_2.bias requires_grad= False
transformer.layer.22.rel_attn.q requires_grad= False
transformer.layer.22.rel_attn.k requires_grad= False
transformer.layer.22.rel_attn.v requires_grad= False
transformer.layer.22.rel_attn.o requires_grad= False
transformer.layer.22.rel_attn.r requires_grad= False
transformer.layer.22.rel_attn.r_r_bias requires_grad= False
transformer.layer.22.rel_attn.r_s_bias requires_grad= False
transformer.layer.22.rel_attn.r_w_bias requires_grad= False
transformer.layer.22.rel_attn.seg_embed requires_grad= False
transformer.layer.22.rel_attn.layer_norm.weight requires_grad= False
transformer.layer.22.rel_attn.layer_norm.bias requires_grad= False
transformer.layer.22.ff.layer_norm.weight requires_grad= False
transformer.layer.22.ff.layer_norm.bias requires_grad= False
transformer.layer.22.ff.layer_1.weight requires_grad= False
transformer.layer.22.ff.layer_1.bias requires_grad= False
transformer.layer.22.ff.layer_2.weight requires_grad= False
transformer.layer.22.ff.layer_2.bias requires_grad= False
transformer.layer.23.rel_attn.q requires_grad= True
transformer.layer.23.rel_attn.k requires_grad= True
transformer.layer.23.rel_attn.v requires_grad= True
transformer.layer.23.rel_attn.o requires_grad= True
transformer.layer.23.rel_attn.r requires_grad= True
transformer.layer.23.rel_attn.r_r_bias requires_grad= True
transformer.layer.23.rel_attn.r_s_bias requires_grad= True
transformer.layer.23.rel_attn.r_w_bias requires_grad= True
transformer.layer.23.rel_attn.seg_embed requires_grad= True

```



```

transformer.layer.23.rel_attn.layer_norm.weight requires_grad= True
transformer.layer.23.rel_attn.layer_norm.bias requires_grad= True
transformer.layer.23.ff.layer_norm.weight requires_grad= True
transformer.layer.23.ff.layer_norm.bias requires_grad= True
transformer.layer.23.ff.layer_1.weight requires_grad= True
transformer.layer.23.ff.layer_1.bias requires_grad= True
transformer.layer.23.ff.layer_2.weight requires_grad= True
transformer.layer.23.ff.layer_2.bias requires_grad= True
sequence_summary.summary.weight requires_grad= True
sequence_summary.summary.bias requires_grad= True
logits_proj.weight requires_grad= True
logits_proj.bias requires_grad= True

```

```

[38]: # 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-20-0816cb2c5003>:31: FutureWarning: `tokenizer` is deprecated and
will be removed in version 5.0.0 for `Trainer.__init__`. Use `processing_class`
instead.
  trainer = Trainer(
model.safetensors:   0%|          | 0.00/1.44G [00:00<?, ?B/s]
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
Validation metrics: {'eval_loss': 0.676538348197937, 'eval_accuracy':
0.6060606060606061, 'eval_precision': 0.5857142857142857, 'eval_recall':
0.803921568627451, 'eval_f1': 0.6776859504132231, 'eval_runtime': 5.4585,
'eval_samples_per_second': 18.137, 'eval_steps_per_second': 0.183, 'epoch': 1.0}
Test metrics: {'eval_loss': 0.6945560574531555, 'eval_accuracy':
0.5054347826086957, 'eval_precision': 0.54, 'eval_recall': 0.5454545454545454,

```

```
'eval_f1': 0.542713567839196, 'eval_runtime': 6.3016, 'eval_samples_per_second': 29.199, 'eval_steps_per_second': 0.317, 'epoch': 1.0}
```

```
[39]: # 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\_xlnet/xlnet-large-cased\_binary\_complexity\_2025-04-11T09:09:22.857081-07:00

<IPython.core.display.HTML object>

EXPERIMENT LOGGED TO:  
/content/drive/MyDrive/266-final/results/experiment\_runs.txt