Paper reproduction: Bringing At-home Pediatric Sleep Apnea Tesing Closer to Reality: A Multi-Modal Transformer Approach

Aaron Schlesinger and Dave Pankros

This notebook is a draft submission of our paper reproduction for CS 598 - Deep Learning for healthcare. In it, we aim to reproduce the findings in the paper *Bringing At-home Pediatric Sleep Apnea Testing Closer to Reality: A Multi-Modal Transformer Approach* [1].

The code herein is functional, with some limitations around data, discussed below. The public GitHub repository in which we do our work and from which we derived this notebook is located at github.com/arschles/UIUC-CS598-DLH-Project. The code therein is organized as a standard Python project, and the repository contains comprehensive instructions on running it.

Executive Summary

Our work to get to this point has been extensive and varied. In the subsequent sections, you will see extensive prose and code detailing the work we've done, but below is a high-level summary, in bulleted form:

- We have acquired a subset of the data used in the paper
- We have completed data preprocessing and data loading
- We have ensured the model trains sufficiently
- We have done standard evaluations of our trained model

From here, we plan to acquire as much additional data as we can (this is not under our control, as described below), clean up the code, perform ablation studies, and generate graphs and other visuals as necessary to support and illustrate our evaluations.

License

The data used by this dataset is subject to a license. While we are not sharing the

original data in its raw form, we are sharing derivations of that data like pre-trained model checkpoints. To facilitate the goals of this assignment and class and to honor the spirit of the license, we require that by opening and running this notebook you agree that you:

1. Will not attempt to reverse engineer the data into a form other than provided,
2. Will not attempt to identify or de-anonymize any individual or institution in the dataset,
3. Will not share or re-use the data in any form,
4. Will not use the data for any purpose other than this assignment, and
5. Maintain a up-to-date cerification in human research subject protection and HIPAA regulations.

The data license is discussed in more detail below.

Preface

Much effort went into making the original paper repository runnable. The repository was lacking any documentation including even python version and package versions. It was built exclusively to be run on Windows and lacked any attempt at cross-platform support. We ran into several cases where the code, as supplied, could never have been run in the provided state. In one instance, for example, the arguments to a function were illegal and, after reaching out to the author and receiving no reply, we used our best judgment at a solution.

With that in mind, we are attempting to reproduce results consistent with the original paper, but there may be differences due to these changes or other instances of errors or omissions that did not cause a code failure and that may have been too subtle to be caught in our initial passes over the code. These differences will, inevitably, cause deviations between our results and the results presented in the paper.

We have, however, undertaken in our fork of the original code to provide more information to aid reproducibility (including a `requirements.txt` file and other detailed dependency information), made the code cross-platform where it was not, and provide for information about how to preprocess and run the code using standard tools like bash and make.

Introduction

Sleep apnea in children is a major health problem that affects between one to five percent of US children, but differs from sleep apnea in adults in its clinical causes and characteristics. Thus, previously-created methods for detecting adult sleep apnea may be ineffective at detecting pediatric sleep apnea.

Background

While there are numerous testing tools and algorithmic methods for detecting adult sleep apnea, the same tools and methods are unavailable for pediatric sleep apnea (PSA) due to these differences. Detecting pediatric sleep apnea more quickly and easily can lead to earlier clinical intervention in the child's care and ultimately prevent the wide variety of health issues commonly caused by OSA. Polysomnography (PSG) is the standard method for formal sleep apnea diagnosis, but is generally performed in a dedicated facility where a patient can be monitored overnight. Polysomnography involves collecting various continuous-time signals, including electroencephalogram (EEG), electrooculogram (EOG), electrocardiogram (ECG), pulse oximetry (SpO2), end-tidal carbon dioxide (ETCO2), respiratory inductance plethysmography (RIP), nasal airflow, and oral airflow. While effective, PSG is, however, complex, costly and requires a dedicated sleep lab.

State of the Art

Current methods target adults and, for reasons stated earlier, are ineffective at diagnosing PSA in childen. Very little work has been done in the scope of pediatric sleep apnea. In general, full Polysomnography data is hard to find and thus, much research has focused on determining the Apnea-Hypopnea Index (AHI) from ECG and SpO2 signals.

While transformers are used commonly in general deep learning models, they are much less prevalent in the detection of sleep apnea. Two studies described in this paper used transformers to determine sleep stages (one in adults, one in children), while another used a hybrid CNN/transformer model of obstructive sleep apnea (OSA) detection.

Paper

The paper proposes to study the gaps in Obstructive Sleep Apnea Hypopnea Syndrome (OSAHS) in children vis-a-vis adults. The paper then suggests a custom transformer-based method and data representation for PSA detection, and identifies the polysomography modalities that most closely correlate to OSAHS in children.

The results presented in the paper portray state-of-the-art results.

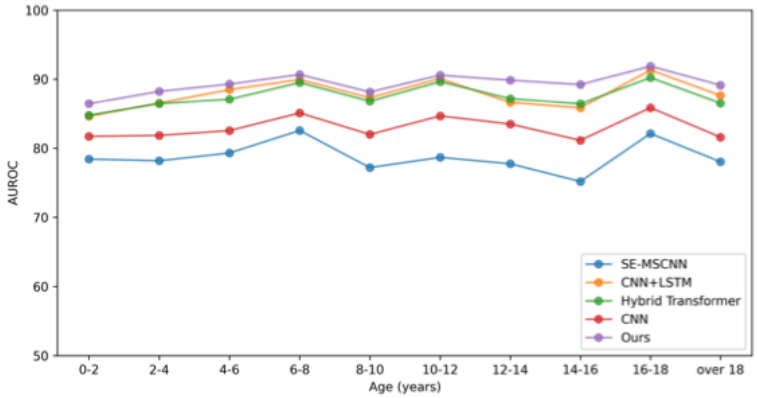


Figure 2: Performance comparison of the proposed and state-of-the-art models on different age groups measured by AUROC on NCH dataset.

If diagnosing pediatric sleep apnea can be done with low-cost, consumer hardware, then the costs of pediatric sleep apena diagnosis decrease and children's health improves. Additionally, lowering the cost of dianoses would also enable access for underserved, including rural, populations with limited access to sleep labs.

Scope of Reproducibility

We will generally investigate whether we can beat state-of-the-art results from Polysomnography (PSG) studies in pediatric sleep apnea detection using the transformer-based model proposed in the paper and discussed above. More specifically, we will focus on testing the following hypotheses:

1. Whether the proposed model can achieve results from signals more easily collected than PSG. As in the paper, we will focus on ECG and SpO_2 signals, and
2. Whether the results support this method being effective as a dedicated method for in-lab sleep studies.

Methodology

In this section, we detail how we acquire, process and load the relevant data, and how we train and evaluate the model given those data. We will exlain the procedure and code in this notebook with the understanding that we did not process the data, train the model, or evaluate the model using this notebook. True reproduction of our environment is explained in the Repository Setup and Evaluation section, immediately following.

Repository Setup

We originally started with the original research code, copied it into this project's repository, and began making changes. As detailed elsewhere, much work was done to get the code to a runnable state, then more was done to make it performant, able to load models from checkpoints, and so on.

When we got to a good state in the repository, we moved code into this notebook and modified it as necessary to work in this environment. Some of those modifications include the following:

- Intra-project imports don't work so we had to remove them and linearize the intra-project dependency graph herein
- Everything is in a global namespace, so we had to fix the naming conflicts that arose herein
- Testing and training code is combined in the repository, but it made more sense to split the two pieces herein, so we had to extract large segments of code common to both operations

As a result of these notebook-specific modifications, the code in both this notebook and our project repository achieves the same result, but is different.

If you'd like to run the code in the repository, please see the `original/HOW_TO_RUN.md` file

Data

Data Licensing Note

Due to the license required to access the data, we cannot provide the raw dataset with this notebook because doing so would be a violation of terms 3 and 4 of the PhysioNetCredential Health Data License 1.5.0.

Because we cannot supply original raw data, we will discuss the processing of the data and the relevant processing code with example output, but the provided code **will not actually process data** within this notebook. Since it is not subject to the aforementioned license, we have stored preprocessed data elsewhere and will load it in relevant sections of this notebook. From that point, further processing will be performed in this notebook.

> Before we proceed, we want to stress that this preprocessed data -- and the model checkpoint data we'll discuss later -- is derived from licensed data for which one must pass a training course to access. This notebook includes instructions and code to access these resources. **By proceeding past this section, you must agree to adhere to the requirements outlined in the license section above**.

Source(s)

The original paper used Nationwide Children's Hospital (NCH) Sleep Data Bank (Lee et al., 2022), and Childhood Adenotonsillectomy Trial (CHAT) dataset (Marcus et al., 2013; Redline et al., 2011). Each of these datasets are collected from actual sleep studies, anonymized and made available for research. At the time of this writing, we have been unable to gain approval to access the CHAT dataset so are unable to use it. We currently have access to the NCH dataset through physionet.org. While we have been downloading this data for over a week, download speeds are capped at around 600KB/s. We have currently downloaded around 400GB of 2.1TB total.

Dataset summary

Since the paper provides extensive statistics on the data used in this study, we have not undertaken to perform our own calculations. The paper-provided measures are as follows:

Demographics of the Datasets

		NCH	CHAT
	Number of Patients	3673	453
	Number of Sleep Studies	3984	453
Sex			
	Male	2068	219
	Female	1604	234
Race			
	Asian	93	8
	Black	738	252
	White	2433	161

Other	409	32
Age (years/mean)	[0-30]/8.8	[5-9]/6.5

Data Statistics

Event	NCH	CHAT
Oxygen Desaturation	215280	65006
Oximeter Event	161641	9,864
EEG arousal	146052	--
Respiratory Events		
Hypopnea	14522	15871
Obstructive Hypopnea	42179	--
Obstructive apnea	15782	7075
Central apnea	6938	3656
Mixed apnea	2650	--
Sleep Stages		
Wake	665676	10282
N1	128410	13578
N2	1383765	19985
N3	875486	9981
REM	611320	3283

Data preprocessing

Data Normalization: Interpolation, Resampling and Tokenization

The raw data consist of 1) regular time-series, 2) irregular time-teries, and 3) tabular data. Additionally, the time-series data may be provided in various frequencies. To merge all the different data types into a coherent dataset suitable for training and testing, the following steps must be performed:

1. Each irregular time series must be interpolated to convert it into a regular time series.

2. All the time series data must be resampled into a uniform frequency, $f_{sampling}$, for all sleep studies and modalities.
3. The tabular data is added to the time series as a constant signal (i.e. repeated tokens)
4. The combined data is split into i equal-length tokens of time S, where each modality consists of $S * f_{sampling}$ data points

This data can then be split and passed to the model for training and testing.

Splitting

This paper utilizes a custom stratified k-fold cross validation to ensure 1) an equal number of patients are assigned to each fold, and to 2) normalize the number of positive samples in each fold. Pseudocode of this method is:

Input: $\{P_n\}_{n=1}^{N}$
Output: $\{fold_f\}_{n=1}^{K}$
for $n \leftarrow 1$ ***to*** N **do**
  $P_i.score = 0$
  for $m \leftarrow 1$ ***to*** M_n **do**
    for $l \leftarrow 1$ ***to*** $L_{n,m}$ **do**
      $P_i.score \mathrel{+}= length(E_{i,j}^{k})$
    end
  end
end
score_sorted_list $\leftarrow$ sort patients by score
for $i \leftarrow 1$ ***to*** N **do**
  $f = i \mod K$
  $fold_f \leftarrow$ score_sorted_list[i]
end

The Apnea-Hypopnea index

The Apnea-Hypopnea index (AHI) is an important quantification of the severity of sleep apnea. Its derivation for a single night of sleep is as follows:

$$\frac{N_{apneic} + N_{hypopneic}}{H}$$

Where N_{apneic} is the number of apneic events (instances where the person stops breathing), $N_{hypopneic}$ is the number of hypopneic events (instances where the airflow is blocked and the person's breathing becomes more shallow), and H is the total number of hours of sleep for the night.

Since it can be derived from an entire night of sleep, the AHI is a very useful measure to summarize, with relatively modest loss of information, a person's apnea/hypopnea activity in a given night of sleep.

Below we show some setup code followed by the code to create a TSV (tab-separated file) file containing AHI measures for a given sleep study.

In [ ]:
```
import os

# raw data is required for both the preprocessing and data loading step.
# since raw data is licensed, we do not, in this notebook, do these steps
# by default.
#
# if you have access to raw data, download it to your machine, set this
# variable to True and set PHYSIONET_ROOT to the location on disk of your da
# physionet.org directory
SHOULD_PREPROCESS = False
PHYSIONET_ROOT="/root/data/physionet.org"
# this is the location of the preprocessed data file
PREPROCESS_URL="https://dlhproject.sfo3.cdn.digitaloceanspaces.com/nch_30x64

# whether to test from a pretrained model checkpoint. set to False if you
# want to test from the model that is trained in this notebook, True if you
# want to download the checkpoint and test with that.
#
# NOTE: the model will train regardless of the value to which you set this
# constant. it only affects how testing is done
#
# If you want to skip training, set the SKIP_TRAINING flag below to True
TEST_FROM_CHECKPOINT = True
CHECKPOINT_URL="https://dlhproject.sfo3.cdn.digitaloceanspaces.com/weights-2

# this is handy to turn off the training process. it should only be
# set to true if TEST_FROM_CHECKPOINT is also set to True
SKIP_TRAINING = False

if SKIP_TRAINING and not TEST_FROM_CHECKPOINT:
    print(
        "WARNING: SKIP_TRAINING is on and TEST_FROM_CHECKPOINT is off. "
```

```
        "This situation will mean a completely untrained model, and your "
        "evaluation metrics will probably be horrible."
    )

#####
# do not change anything under this line
#####
AHI_OUT_PATH = os.path.join(PHYSIONET_ROOT, 'AHI.csv')
PREPROCESS_OUT_PATH = os.path.join(PHYSIONET_ROOT, 'nch_30x64.npz')
NCH_DATA_ROOT = os.path.join(PHYSIONET_ROOT, 'files', 'nch-sleep', '3.1.0')
HEALTH_DATA_ROOT = os.path.join(NCH_DATA_ROOT, 'Health_Data')
SLEEP_DATA_ROOT = os.path.join(NCH_DATA_ROOT, 'Sleep_Data')
MODEL_OUT_PATH = os.path.join('original', 'weights', 'semscnn_ecgspo2', 'f')
```

In [ ]:

```
import os
import os.path
import pandas as pd
from datetime import datetime
import csv

APNEA_EVENT_DICT = {
    "Obstructive Apnea": 2,
    "Central Apnea": 2,
    "Mixed Apnea": 2,
    "apnea": 2,
    "obstructive apnea": 2,
    "central apnea": 2,
    "apnea": 2,
    "Apnea": 2,
}

HYPOPNEA_EVENT_DICT = {
    "Obstructive Hypopnea": 1,
    "Hypopnea": 1,
    "hypopnea": 1,
    "Mixed Hypopnea": 1,
    "Central Hypopnea": 1,
}

def _num_sleep_hours(
    sleep_study_metadata: pd.DataFrame,
    pat_id: int,
    study_id: int,
) -> int:
    ssm = sleep_study_metadata
    sleep_duration_df = ssm.loc[
        (ssm["STUDY_PAT_ID"] == pat_id) &
        (ssm["SLEEP_STUDY_ID"] == study_id)
    ]
    assert len(sleep_duration_df) == 1, (
```

```
        f'expected just 1 study with patient {pat_id} and study '
        f'{study_id}, but got {len(sleep_duration_df)} instead'
    )
    sleep_duration_datetime = datetime.strptime(
        str(
            sleep_duration_df[
                "SLEEP_STUDY_DURATION_DATETIME"
            ].iloc[0]
        ).strip(),
        "%H:%M:%S"
    )
    return sleep_duration_datetime.hour


def _ahi_for_study(
    sleep_study_metadata: pd.DataFrame,
    sleep_study: pd.DataFrame,
    pat_id: int,
    study_id: int,
) -> float:
    '''
    Calculate the apnea-hypopnea index (AHI) for a given sleep study.
    All apnea and hypopnea events will be counted from the sleep_study
    DataFrame, and then divided by the total sleep duration, which
    will be gotten from the sleep_study_metadata DataFrame. The result will
    be returned as a float

    For more on AHI, see the following link:

    https://www.sleepfoundation.org/sleep-apnea/ahi

    :param sleep_study_metadata
        the DataFrame that has at least the following columns in order
        from left to right:
        STUDY_PAT_ID,
        SLEEP_STUDY_ID,
        SLEEP_STUDY_START_DATETIME,
        SLEEP_STUDY_DURATION_DATETIME
    :param sleep_study
        the data from the sleep study in which we're interested
    :param pat_id
        the ID of the patient on whom the given study was done
    :param study_id
        the ID of the study

    :return
        the AHI for the given study, as a float value
    '''

    # example tsv file:
```

```
    # onset duration description
    # 29766.7421875     11.0546875      Obstructive Hypopnea

    df = sleep_study
    hypopnea_keys = set(HYPOPNEA_EVENT_DICT.keys())
    apnea_keys = set(APNEA_EVENT_DICT.keys())

    hypopnea_events = df.loc[df["description"].isin(hypopnea_keys)]
    apnea_events = df.loc[df["description"].isin(apnea_keys)]
    total_num_events = len(hypopnea_events) + len(apnea_events)
    sleep_hours = float(_num_sleep_hours(
        sleep_study_metadata,
        pat_id,
        study_id,
    ))
    return float(total_num_events) / sleep_hours


def _parse_ss_tsv_filename(filename: str) -> tuple[int, int]:
    '''
    given a sleep study filename like `10048_24622.tsv`, that represents
    <patient_id>_<sleep_study_id>.tsv, return a 2-tuple containing
    the patient ID in element 1 and sleep study ID in element 2
    '''
    if not filename.endswith(".tsv"):
        raise FileNotFoundError(
            f"expected {filename} to end with .tsv but it didn't"
        )

    underscore_spl = filename.split("/")[-1][:-4].split("_")
    if len(underscore_spl) != 2:
        raise FileNotFoundError(f'malformed filename {filename}')
    [pat_id, study_id] = underscore_spl
    return (int(pat_id), int(study_id))


def _write_tsv(out_filename: str, data: list[tuple[str, str, float]]):
    with open(out_filename, 'w', newline='') as tsvfile:
        writer = csv.writer(tsvfile, delimiter=',')
        # in ./preprocessing.py, we need to have at least 'Study'
        # and 'AHI'. Since they chose PascalCase, I extended that usage
        # to patient ID.
        writer.writerow(("PatID", "Study", "AHI"))
        for row in data:
            writer.writerow(row)


def calculate_ahi(
    sleep_study_metadata_file: str,
    sleep_study_root: str,
```

```
    out_file: str,
) -> None:
    '''
    Calculate the AHI values from a sleep study's metadata file and all the
    sleep measurements in a given directory. See the _ahi_for_study functior
    for an overview of how the metadata file and sleep measurement files sho
    be structured.

    :param sleep_study_metadata_file - the metadata file summarizing the sle
        study
    :param sleep_study_root - the root directory containing all the individu
        sleep study measures

    :return out_file - the name of the file to which to write the AHI values
        in TSV format

    '''
    metadata_df = pd.read_csv(
        sleep_study_metadata_file,
        sep=","
    )

    tsv_files = [
        f for f in os.listdir(sleep_study_root)
        if f.endswith(".tsv")
    ]

    print(
        f"creating AHI from {len(tsv_files)} sleep studies in "
        f"{sleep_study_root} and outputting the AHI results to {out_file}"
    )

    # each tuple is (patient_id, study_id, AHI)
    results: list[tuple[str, str, float]] = []
    for tsv_file in tsv_files:
        filename = os.path.join(sleep_study_root, tsv_file)
        pat_id, study_id = _parse_ss_tsv_filename(filename)
        sleep_study_df = pd.read_csv(
            filename,
            sep="\t",
        )
        ahi = _ahi_for_study(
            metadata_df,
            sleep_study_df,
            pat_id,
            study_id,
        )
        results.append((pat_id, study_id, ahi))
    _write_tsv(out_file, results)
```

```
def generate_ahi_file(data_root: str, out_file: str) -> None:
    sleep_study_metadata_file = os.path.join(
        data_root,
        "files",
        "nch-sleep",
        "3.1.0",
        "Health_Data",
        "SLEEP_STUDY.csv"
    )
    sleep_study_root = os.path.join(
        data_root,
        "files",
        "nch-sleep",
        "3.1.0",
        "Sleep_Data"
    )

    calculate_ahi(
        sleep_study_metadata_file,
        sleep_study_root,
        out_file=out_file,
    )

if SHOULD_PREPROCESS:
    print(
        f"SHOULD_PREPROCESS is true, so generating AHI file from data at "
        f"{PHYSIONET_ROOT} and outputting to {AHI_OUT_PATH}"
    )
    generate_ahi_file(PHYSIONET_ROOT, AHI_OUT_PATH)
```

Preprocessing code

With the AHI values calculated and saved to a TSV file, we can move onto data preprocessing. Roughly speaking, the goals of preprocessing are as follows for each sleep study:

- Ignore the study if its AHI is lower than a threshold
- Collate and pad as necessary the relevant sleep events in the study
- Resample samples from raw data as necessary
  - The motivation behind, and method for Resampling was discussed above
- Write results to a compressed numpy representation

Sleep study loading and manipulation logic

First, we have foundational logic for loading sleep study and health data files.

In [ ]:
```
import os
import pandas as pd

def init_study_list():
    return [x[:-4] for x in os.listdir(SLEEP_DATA_ROOT) if x.endswith('edf')

def init_age_file():
    new_fn = 'age_file.csv'
    age_path = os.path.join(HEALTH_DATA_ROOT, 'SLEEP_STUDY.csv')

    df = pd.read_csv(age_path, sep=',', dtype='str')
    df['FILE_NAME'] = df["STUDY_PAT_ID"].str.cat(df["SLEEP_STUDY_ID"], sep='

    df.to_csv(new_fn, columns=["FILE_NAME", "AGE_AT_SLEEP_STUDY_DAYS"], inde
    return os.path.abspath(new_fn)

def load_health_info(name: str, convert_datetime: bool = True):
    assert type(name) == str

    path = os.path.join(HEALTH_DATA_ROOT, name)
    df = pd.read_csv(path)

    if convert_datetime:
        if name == DEMOGRAPHIC:
            df['BIRTH_DATE'] = pd.to_datetime(df['BIRTH_DATE'], infer_dateti
        elif name == DIAGNOSIS:
            df['DX_START_DATETIME'] = pd.to_datetime(df['DX_START_DATETIME']
            df['DX_END_DATETIME'] = pd.to_datetime(df['DX_END_DATETIME'], ir
        elif name == ENCOUNTER:
            df['ENCOUNTER_DATE'] = pd.to_datetime(df['ENCOUNTER_DATE'], infe
            df['VISIT_START_DATETIME'] = pd.to_datetime(df['VISIT_START_DATE
            df['VISIT_END_DATETIME'] = pd.to_datetime(df['VISIT_END_DATETIME
            df['ADT_ARRIVAL_DATETIME'] = pd.to_datetime(df['ADT_ARRIVAL_DATE
            df['ED_DEPARTURE_DATETIME'] = pd.to_datetime(df['ED_DEPARTURE_DA
        elif name == MEASUREMENT:
            df['MEAS_RECORDED_DATETIME'] = pd.to_datetime(df['MEAS_RECORDED_
        elif name == MEDICATION:
            df['MED_START_DATETIME'] = pd.to_datetime(df['MED_START_DATETIME
            df['MED_END_DATETIME'] = pd.to_datetime(df['MED_END_DATETIME'],
            df['MED_ORDER_DATETIME'] = pd.to_datetime(df['MED_ORDER_DATETIME
            df['MED_TAKEN_DATETIME'] = pd.to_datetime(df['MED_TAKEN_DATETIME
        elif name == PROCEDURE:
            df['PROCEDURE_DATETIME'] = pd.to_datetime(df['PROCEDURE_DATETIME
        elif name == PROCEDURE_SURG_HX:
            df['PROC_NOTED_DATE'] = pd.to_datetime(df['PROC_NOTED_DATE'], ir
            df['PROC_START_TIME'] = pd.to_datetime(df['PROC_START_TIME'], ir
            df['PROC_END_TIME'] = pd.to_datetime(df['PROC_END_TIME'], infer_
        elif name == SLEEP_STUDY_NAME:
            df['SLEEP_STUDY_START_DATETIME'] = pd.to_datetime(df['SLEEP_STUD
```

```
    return df

if SHOULD_PREPROCESS:
    print(f'SHOULD_PREPROCESS is true, so loading sleep study information ir
    ss_study_list = init_study_list()
    age_fn = init_age_file()
    SLEEP_STUDY = load_health_info("SLEEP_STUDY.csv", False)
```

Preprocessing code

Next, the actual preprocessing code.

In [ ]:
```
import concurrent.futures
import os.path
import sys
from datetime import datetime
import pandas as pd
import mne
import numpy as np
from biosppy.signals.ecg import hamilton_segmenter, correct_rpeaks
from biosppy.signals import tools as st
from mne import make_fixed_length_events
from scipy.interpolate import splev, splrep
from itertools import compress

mne.set_log_file('log.txt', overwrite=False)

CHUNK_DURATION = 30.0
FREQ = 64.0

POS_EVENT_DICT: dict[str, int] = {
    "Obstructive Hypopnea": 1,
    "Hypopnea": 1,
    "hypopnea": 1,
    "Mixed Hypopnea": 1,
    "Central Hypopnea": 1,

    "Obstructive Apnea": 2,
    "Central Apnea": 2,
    "Mixed Apnea": 2,
    "apnea": 2,
    "obstructive apnea": 2,
    "central apnea": 2,
    "Apnea": 2,
}

WAKE_DICT: dict[str, int] = {
    "Sleep stage W": 10
```

```
}

########################################## Annotation Modifier functions ###
def identity(df):
    return df


def apnea2bad(df):
    df = df.replace(r'.*pnea.*', 'badevent', regex=True)
    print("bad replaced!")
    return df


def wake2bad(df):
    return df.replace("Sleep stage W", 'badevent')


def change_duration(df, label_dict=POS_EVENT_DICT, duration=CHUNK_DURATION):
    for key in label_dict:
        df.loc[df.description == key, 'duration'] = duration
    print("change duration!")
    return df

def preprocess(i, annotation_modifier, out_dir, ahi_dict):
    is_apnea_available, is_hypopnea_available = True, True
    study = ss.data.study_list[i]

    # print(f"loading study {study}")
    raw = ss.data.load_study(study, annotation_modifier, verbose=True)

    ########################################   CHECK CRITERIA FOR SS   #####
    if not all([name in raw.ch_names for name in channels]):
        print("study " + str(study) + " skipped since insufficient channels"
        return 0

    ahi_value = ahi_dict.get(study,None)
    if ahi_value is None:
        print(ahi_dict)
        print("study " + str(study) + " skipped since AHI is MISSING.  Is AH
        return 0

    if ahi_value < THRESHOLD:
        print("study " + str(study) + " skipped since low AHI ---  AHI = " +
        return 0

    try:
        apnea_events, event_ids = mne.events_from_annotations(
            raw,
            event_id=POS_EVENT_DICT,
            chunk_duration=1.0,
```

```
            verbose=None
        )
        # print('|')
    except ValueError:
        print("No Chunk found!", file=sys.stderr)
        return 0
    except Exception as e:
        print(e)
        return 0
    ########################################   CHECK CRITERIA FOR SS   #####
    print(str(i) + "---" + str(datetime.now().time().strftime("%H:%M:%S")) +

    try:
        apnea_events, event_ids = mne.events_from_annotations(
            raw,
            event_id=APNEA_EVENT_DICT,
            chunk_duration=1.0,
            verbose=None
        )
    except ValueError:
        is_apnea_available = False

    try:
        hypopnea_events, event_ids = mne.events_from_annotations(
            raw,
            event_id=HYPOPNEA_EVENT_DICT,
            chunk_duration=1.0,
            verbose=None
        )
    except ValueError:
        is_hypopnea_available = False

    wake_events, event_ids = mne.events_from_annotations(
        raw,
        event_id=WAKE_DICT,
        chunk_duration=1.0,
        verbose=None
    )
    ##########################################################################
    sfreq = raw.info['sfreq']
    tmax = CHUNK_DURATION - 1. / sfreq

    raw = raw.pick_channels(channels, ordered=True)
    fixed_events = make_fixed_length_events(
        raw,
        id=0,
        duration=CHUNK_DURATION,
        overlap=0.
    )
    epochs = mne.Epochs(
```

```
        raw,
        fixed_events,
        event_id=[0],
        tmin=0,
        tmax=tmax,
        baseline=None,
        preload=True,
        proj=False,
        verbose=None
    )
    epochs.load_data()
    if sfreq != FREQ:
        epochs = epochs.resample(FREQ, npad='auto', n_jobs=4, verbose=None)
    data = epochs.get_data()
    ##########################################################################
    if is_apnea_available:
        apnea_events_set = set((apnea_events[:, 0] / sfreq).astype(int))
    if is_hypopnea_available:
        hypopnea_events_set = set((hypopnea_events[:, 0] / sfreq).astype(int
    wake_events_set = set((wake_events[:, 0] / sfreq).astype(int))

    starts = (epochs.events[:, 0] / sfreq).astype(int)

    labels_apnea = []
    labels_hypopnea = []
    labels_wake = []
    total_apnea_event_second = 0
    total_hypopnea_event_second = 0

    for seq in range(data.shape[0]):
        epoch_set = set(range(starts[seq], starts[seq] + int(CHUNK_DURATION)
        if is_apnea_available:
            apnea_seconds = len(apnea_events_set.intersection(epoch_set))
            total_apnea_event_second += apnea_seconds
            labels_apnea.append(apnea_seconds)
        else:
            labels_apnea.append(0)

        if is_hypopnea_available:
            hypopnea_seconds = len(hypopnea_events_set.intersection(epoch_se
            total_hypopnea_event_second += hypopnea_seconds
            labels_hypopnea.append(hypopnea_seconds)
        else:
            labels_hypopnea.append(0)

        labels_wake.append(len(wake_events_set.intersection(epoch_set)) == 0
    ##########################################################################
    print(study + "    HAMED    " + str(len(labels_wake) - sum(labels_wake))
    data = data[labels_wake, :, :]
    labels_apnea = list(compress(labels_apnea, labels_wake))
```

```
    labels_hypopnea = list(compress(labels_hypopnea, labels_wake))

    out_name = study + "_" + str(total_apnea_event_second) + "_" + str(total
    out_path = os.path.join(out_dir, out_name)
    # print(f"Saving {study} to {out_path}.npz")
    np.savez_compressed(out_path, data=data, labels_apnea=labels_apnea, labe

    return data.shape[0]

def run_preprocess(
    ahi_path: str,
    out_folder: str,
    n_studies: int = 3984,
    n_workers: int = 3
):
    if not os.path.exists(ahi_path):
        return FileNotFoundError(f'AHI file {ahi_path} was not found!')

    ahi = pd.read_csv(ahi_path)
    # filename is <patient_id>_<study>
    filenames = ahi['PatID'].astype(str) + '_' + ahi['Study'].astype(str)
    # ahi_dict = dict(zip(ahi.Study, ahi.AHI))
    ahi_dict = dict(zip(filenames, ahi['AHI']))

    if not os.path.exists(out_folder):
        os.mkdir(out_folder)

    if n_workers < 2:
        for idx in range(n_studies):
            preprocess(
                ahi_path=idx,
                out_folder=identity,
                n_studies=out_folder,
                n_workers=ahi_dict
            )
    else:
        with concurrent.futures.ThreadPoolExecutor(
            max_workers=n_workers
        ) as executor:
            executor.map(
                preprocess,
                range(n_studies),
                [identity] * n_studies,
                [out_folder] * n_studies,
                [ahi_dict] * n_studies
            )

if SHOULD_PREPROCESS:
    print(f'SHOULD_PREPROCESS is true, so running preprocessing logic')
    run_preprocess(AHI_OUT_PATH, PREPROCESS_OUT_PATH)
```

Data Loading

After data are preprocessed, we are left with a compressed numpy array file, also called an `npz` file. At this point, we are ready to take the final step before model training - data loading.

The data loader code requires familiar collation and padding logic shown below. Notably, we make use of TensorFlow's ragged tensors to simply and easily handle the task of padding.

In [ ]:
```python
from typing import Any
import numpy.typing as npt
import tensorflow as tf
import tensorflow.ragged

def max_dimensions(
        lst: list[Any],
        level: int=0,
        max_dims: list[int] | None = None
) -> tuple[int, ...]:
    """
    Finds the maximum dimension for each level of a nested list structure

    :param lst: The list for which to get dimensions
    :param level: INTERNAL USE ONLY (the dimension we are processing)
    :param max_dims: INTERNAL USE ONLY (the current array of maximums)
    :return: a tuple of sizes, similar to torch.Tensor.shape()
    """
    if max_dims is None:
        max_dims = []

    # Extend the max_dims list if this is the deepest level we've encountere
    if level >= len(max_dims):
        max_dims.append(len(lst))
    else:
        max_dims[level] = max(max_dims[level], len(lst))

    for item in lst:
        if isinstance(item, list):
            # Recursively process each sublist
            max_dimensions(item, level + 1, max_dims)

    return tuple(max_dims)

def pad_lists(
    lst: list[Any],
```

```
    pad_with: int = 0
) -> npt.NDArray:
    """
    Given a ragged nested list structure `lst` (i.e. where the length
    in each dimension is not uniform), return a new list with all
    levels of the list padded to the maximal length of any list in that
    dimension. Padding elements will have the same value as given in
    `pad_with`

    For example, the return value of `pad_lists([[1], [1, 2]], 0)` will
    be `[[1, 0], [1, 2]]`

    :param lst: the list to pad, if it is ragged. if it's not, this functior
        is a no-op
    :param pad_with: the value to use for padding
    """
    # max_dims[0] is the number of elements (either lists or ints) we
    # need in this dimension
    assert type(lst) is list
    rt = tensorflow.ragged.constant(lst)
    return rt.to_tensor(pad_with).numpy()
```

Then, with padding and collating handled, data loading code is as follows:

In [ ]:
```
import glob
import os
import random
import sys
from typing import Any
from urllib.request import urlretrieve

import numpy as np
import pandas as pd
from scipy.signal import resample
from biosppy.signals.ecg import hamilton_segmenter, correct_rpeaks
from biosppy.signals import tools as st
from scipy.interpolate import splev, splrep

# "EOG LOC-M2",  # 0
# "EOG ROC-M1",  # 1
# "EEG C3-M2",  # 2
# "EEG C4-M1",  # 3
# "ECG EKG2-EKG",  # 4
#
# "RESP PTAF",  # 5
# "RESP AIRFLOW",  # 6
# "RESP THORACIC",  # 7
# "RESP ABDOMINAL",  # 8
# "SPO2",  # 9
```

```
# "CAPNO",  # 10

######### ADDED IN THIS STEP #########
# RRI #11
# Ramp #12
# Demo #13


SIGS = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
s_count = len(SIGS)

THRESHOLD = 3
FREQ = 64
EPOCH_DURATION = 30
ECG_SIG = 4

AHI_PATH = AHI_OUT_PATH
OUT_PATH = PREPROCESS_OUT_PATH
PATH=os.path.join(PHYSIONET_ROOT, "nch_30x64")


def extract_rri(signal, ir, CHUNK_DURATION):
    tm = np.arange(0, CHUNK_DURATION, step=1 / float(ir))  # TIME METRIC FOR

    # print('filtering', signal, FREQ)
    # TODO: Temporarily bypassed until we know how we want to handle this
    # filtered, _, _ = st.filter_signal(signal=signal, ftype="FIR", band="ba
    #                                   frequency=[3, 45], sampling_rate=FRE
    filtered, _, _ = st.filter_signal(signal=signal, ftype="FIR", band="banc
                                      frequency=[3, 30], sampling_rate=FREQ,
    (rpeaks,) = hamilton_segmenter(signal=filtered, sampling_rate=FREQ)
    (rpeaks,) = correct_rpeaks(signal=filtered, rpeaks=rpeaks, sampling_rate

    if 4 < len(rpeaks) < 200:  # and np.max(signal) < 0.0015 and np.min(sign
        rri_tm, rri_signal = rpeaks[1:] / float(FREQ), np.diff(rpeaks) / flo
        ampl_tm, ampl_signal = rpeaks / float(FREQ), signal[rpeaks]
        rri_interp_signal = splev(tm, splrep(rri_tm, rri_signal, k=3), ext=1
        amp_interp_signal = splev(tm, splrep(ampl_tm, ampl_signal, k=3), ext

        return np.clip(rri_interp_signal, 0, 2), np.clip(amp_interp_signal,
    else:
        return np.zeros((FREQ * EPOCH_DURATION)), np.zeros((FREQ * EPOCH_DUR


def load_data(path: str) -> tuple[list[Any], list[Any], list[Any]]:
    '''
    given a path to the directory of sleep studies
    (i.e. DATA_ROOT/physionet.org/nch_30x64), load and process all sleep
    studies, then return them
    '''
```

```
    # demo = pd.read_csv("../misc/result.csv") # TODO

    ahi = pd.read_csv(AHI_PATH)
    filename = ahi.PatID.astype(str) + '_' + ahi.Study.astype(str)
    ahi_dict = dict(zip(filename, ahi.AHI))
    root_dir = os.path.expanduser(path)
    file_list = os.listdir(root_dir)
    length = len(file_list)

    print(f"Using AHI from {AHI_PATH}")
    print(f"Using npz files from {root_dir}")
    print(f"Files {file_list}")

    study_event_counts = {}
    apnea_event_counts = {}
    hypopnea_event_counts = {}
    ######################################## Count the respiratory events ##
    for i in range(length):
        # skip directories
        if os.path.isdir(file_list[i]):
            continue

        # print(f"Processing {file_list[i]}")
        try:
            # parts = file_list[i].split("_")
            # parts[0] should be nch

            patient_id = (file_list[i].split("_")[0])
            study_id = (file_list[i].split("_")[1])
            apnea_count = int((file_list[i].split("_")[2]))
            hypopnea_count = int((file_list[i].split("_")[3]).split(".")[0])
        except Exception as e:
            print(f"Filename mismatch. Skipping {file_list[i]} ({e})", file=
            continue
        filename = f"{patient_id}_{study_id}"
        ahi_value = ahi_dict.get(filename, None)
        if ahi_value is None:
            print(f"Sleep study {filename} is not found in AHI.csv.  Skippir
            continue

        try:
            if ahi_value > THRESHOLD:
                apnea_event_counts[patient_id] = apnea_event_counts.get(pati
                hypopnea_event_counts[patient_id] = hypopnea_event_counts.ge
                study_event_counts[patient_id] = study_event_counts.get(pati
        except Exception as e:
            print(f"File structure problem.  Skipping {file_list[i]} ({e})",
            continue

    apnea_event_counts = sorted(apnea_event_counts.items(), key=lambda item:
```

```
    hypopnea_event_counts = sorted(hypopnea_event_counts.items(), key=lambda
    study_event_counts = sorted(study_event_counts.items(), key=lambda item:

    ################################### Fold the data based on number of res
    folds = []
    for i in range(5):
        folds.append(study_event_counts[i::5])

    # print('FOLDS:', folds)

    x = []
    y_apnea = []
    y_hypopnea = []
    counter = 0
    for idx, fold in enumerate(folds):
        first = True
        aggregated_data = None
        aggregated_label_apnea = None
        aggregated_label_hypopnea = None
        for patient in fold:
            counter += 1
            # print(counter)
            glob_path = os.path.join(PATH, patient[0] + "_*")
            # print("glob path", glob_path)
            for study in glob.glob(glob_path):
                study_data = np.load(study)

                signals = study_data['data']
                labels_apnea = study_data['labels_apnea']
                labels_hypopnea = study_data['labels_hypopnea']

                identifier = study.split(os.path.sep)[-1].split('_')[0] + "_
                # print(identifier)
                # demo_arr = demo[demo['id'] == identifier].drop(columns=['i

                y_c = labels_apnea + labels_hypopnea
                neg_samples = np.where(y_c == 0)[0]
                pos_samples = list(np.where(y_c > 0)[0])
                ratio = len(pos_samples) / len(neg_samples)
                neg_survived = []
                for s in range(len(neg_samples)):
                    if random.random() < ratio:
                        neg_survived.append(neg_samples[s])
                samples = neg_survived + pos_samples
                signals = signals[samples, :, :]
                labels_apnea = labels_apnea[samples]
                labels_hypopnea = labels_hypopnea[samples]

                data = np.zeros((signals.shape[0], EPOCH_DURATION * FREQ, s_
                for i in range(signals.shape[0]):  # for each epoch
```

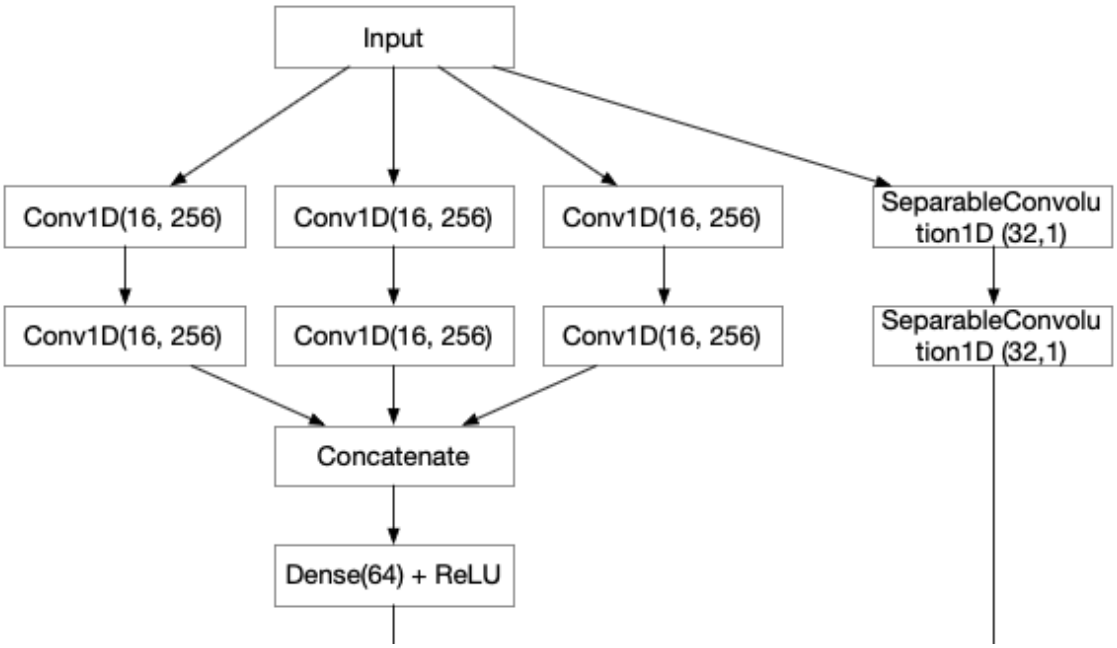
```
                    # data[i, :len(demo_arr), -1] = demo_arr TODO
                    data[i, :, -2], data[i, :, -3] = extract_rri(signals[i,
                    for j in range(s_count):  # for each signal
                        data[i, :, j] = resample(signals[i, SIGS[j], :], EPO

                if first:
                    aggregated_data = data
                    aggregated_label_apnea = labels_apnea
                    aggregated_label_hypopnea = labels_hypopnea
                    first = False
                else:
                    aggregated_data = np.concatenate((aggregated_data, data)
                    aggregated_label_apnea = np.concatenate((aggregated_labe
                    aggregated_label_hypopnea = np.concatenate((aggregated_l

        if aggregated_data is not None:
            x.append(aggregated_data.tolist())
        if aggregated_label_apnea is not None:
            y_apnea.append(aggregated_label_apnea.tolist())
        if aggregated_label_hypopnea is not None:
            y_hypopnea.append(aggregated_label_hypopnea.tolist())

    return x, y_apnea, y_hypopnea

def list_lengths(lst):
    """
    Gets all the individual lengths of a list
    :param lst:
    :return:
    """
    if isinstance(lst, list):
        # For each item in the list, recursively process it if it's a list
        # Otherwise, the item itself is not counted and is represented as No
        sublengths = [list_lengths(item) for item in lst]
        if len([*filter(lambda v: v is not None, sublengths)]) == 0:
            return len(lst)
        # Instead of returning None for non-list items, you could choose to
        return len(lst), sublengths  # Return the length of the current list
    # Return None or some indication for non-list items, if needed
    return None

def run_load_data():
    raw_data_root = os.path.join(PHYSIONET_ROOT, "nch_30x64")
    x, y_apnea, y_hypopnea = load_data(raw_data_root)

    # these output the maximum size for dimension.
    # If we're going to make this a consistent size without truncating,
    # this is the size to make it
    print(f"Padded X.shape:{max_dimensions(x)}")
    x_norm = pad_lists(x, 0)
```

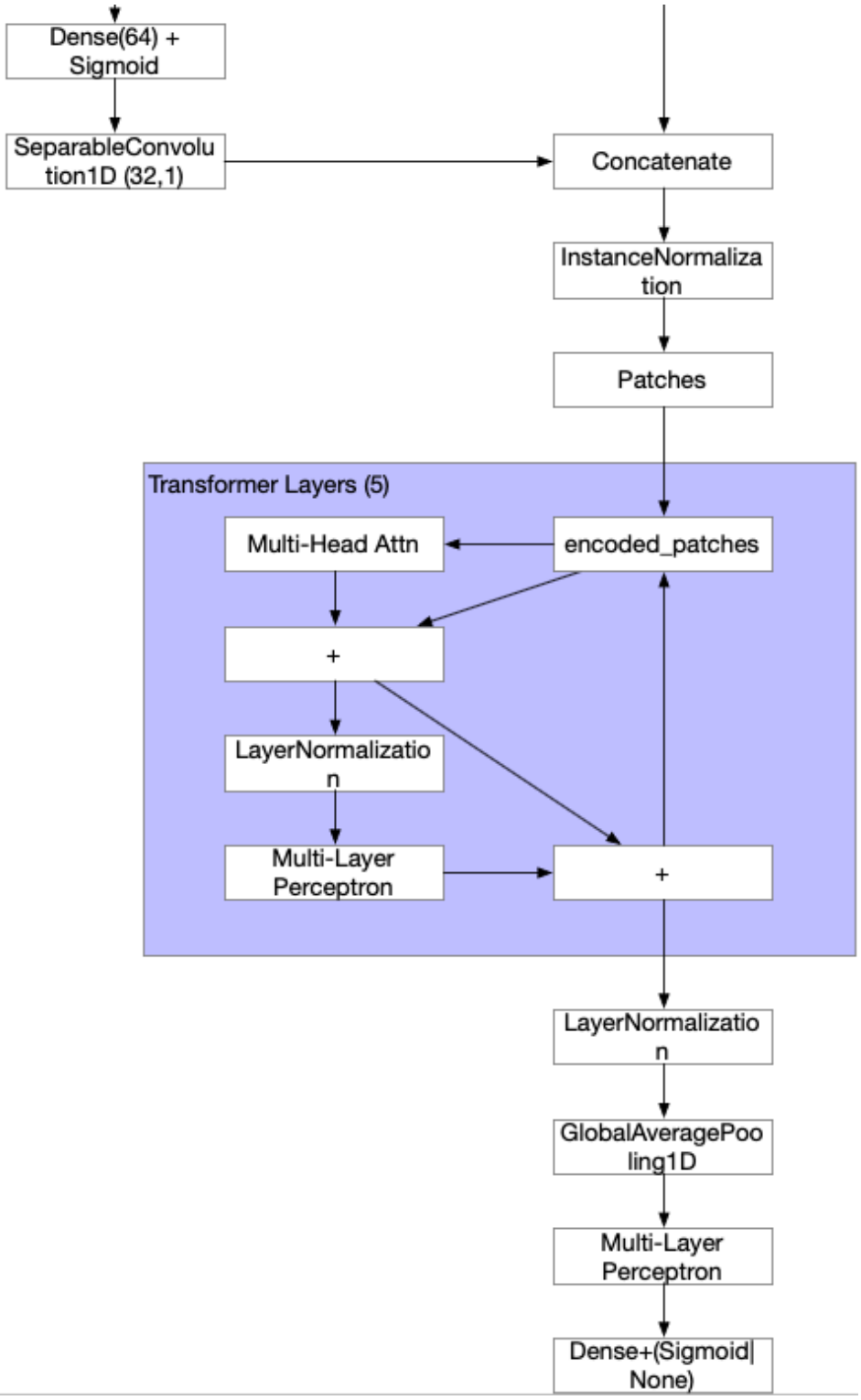
```
    print(f"Padded Y_a shape: {max_dimensions(y_apnea)}")
    y_apnea_norm = pad_lists(y_apnea, 0)

    print(f"Padded Y_h.shape:{max_dimensions(y_hypopnea)}")
    y_hypopnea_norm = pad_lists(y_hypopnea, 0)

    print(f"Saving to {OUT_PATH}")
    np.savez_compressed(
        OUT_PATH,
        x=x_norm,
        y_apnea=y_apnea_norm,
        y_hypopnea=y_hypopnea_norm,
    )

if SHOULD_PREPROCESS:
    print(f'SHOULD_PREPROCESS set to true')
    print(f'PREPROCESS_OUT_PATH={PREPROCESS_OUT_PATH}')
    run_load_data()
```

Model

As discussed above, the model this paper proposes is based on the transformer architecture). While this architecture has led to excellent results in NLP and other applications, it has, to our knowledge, not been studied in the context of sleep apnea.

Like many other transformer-based architectures, this model has several other components, illustrated below.





As seen in this architecture, inputs, which are primarily pre-processed signals data, are first fed in parallel through a variety of 1-D convolutional layers, partially concatenated to fewer parallel "tracks", and then those results are passed through an activation layer. Next, all tracks are concatenated prior to being passed into the transformer, which includes at least one multi-headed attention mechanism and one dense layer. Finally, the

output of the transformer is passed through a normalization layer, pooling layer, fully-connected layer (labeled "Multi-layer perceptron" in the illustration) and a final activation function.

As mentioned in previous sections, we provide a pre-trained model because we are unable to share raw, preprocessed, or loaded data in any form due to licensing issues. Model code is shown in subsequent sections.

Preprocessed data loading

All code prior to this section relies on raw data being present. After running that code, a single compressed numpy arrays file (an `npz` file, for short) is written to disk. Below is code to ensure that npz file is in place and, if not, download it if the `SHOULD_PREPROCESS` flag is set to `False`.

In [ ]:
```
import os

if os.path.exists(PREPROCESS_OUT_PATH):
    # if the file exists, just move on regardless of whether it was generate
    # in this notebook or downloaded
    print(f'preprocessed file found at expected location {PREPROCESS_OUT_PAT
elif SHOULD_PREPROCESS:
    # otherwise, the file was not found but the preprocessing flag was set,
    # so this notebook should have processed it.
    raise FileNotFoundError(
        f'ERROR: preprocessed data at path {PREPROCESS_OUT_PATH} does '
        'not exist. please check that preprocessing succeeded'
    )
else:
    print(
        f'preprocessed file {PREPROCESS_OUT_PATH} does not exist,'
        f'downloading from {PREPROCESS_URL}'
    )
    def dl_report_hook(blcks_sofar: int, blck_size: int, tot_file_size: int)
        print(
            f'({blcks_sofar} blocks downloaded), '
            f'{blcks_sofar*blck_size} of {tot_file_size}'
        )
    urlretrieve(
        url=PREPROCESS_URL,
        filename=PREPROCESS_OUT_PATH,
        reporthook=dl_report_hook,
    )
```

```
preprocessed file found at expected location /root/data/physionet.org/nch_30
x64.npz
```

Transformer layers

Since model code is extensive, we split it up into two sections. We show the transformer model code below:

In [ ]:
```
import keras
import tensorflow as tf
import tensorflow_addons as tfa
from keras import Model
from keras.activations import sigmoid, relu
from keras.layers import Dense, Dropout, Reshape, LayerNormalization, MultiH
    GlobalAveragePooling1D, AveragePooling1D, Concatenate, SeparableConvolut
from keras.regularizers import L2



class Patches(Layer):
    def __init__(self, patch_size):
        super(Patches, self).__init__()
        self.patch_size = patch_size

    def call(self, input):
        input = input[:, tf.newaxis, :, :]
        batch_size = tf.shape(input)[0]
        patches = tf.image.extract_patches(
            images=input,
            sizes=[1, 1, self.patch_size, 1],
            strides=[1, 1, self.patch_size, 1],
            rates=[1, 1, 1, 1],
            padding="VALID",
        )
        patch_dims = patches.shape[-1]
        patches = tf.reshape(patches,
                             [batch_size, -1, patch_dims])
        return patches


class PatchEncoder(Layer):
    def __init__(self, num_patches, projection_dim, l2_weight):
        super(PatchEncoder, self).__init__()
        self.projection_dim = projection_dim
        self.l2_weight = l2_weight
        self.num_patches = num_patches
        self.projection = Dense(units=projection_dim, kernel_regularizer=L2(
                                bias_regularizer=L2(l2_weight))
        self.position_embedding = tf.keras.layers.Embedding(
            input_dim=num_patches, output_dim=projection_dim)
```

```
    def call(self, patch):
        positions = tf.range(start=0, limit=self.num_patches, delta=1)
        encoded = self.projection(patch) # + self.position_embedding(positic
        return encoded


def mlp(x, hidden_units, dropout_rate, l2_weight):
    for _, units in enumerate(hidden_units):
        x = Dense(units, activation=None, kernel_regularizer=L2(l2_weight),
        x = tf.nn.gelu(x)
        x = Dropout(dropout_rate)(x)
    return x


def create_transformer_model(input_shape, num_patches,
                             projection_dim, transformer_layers,
                             num_heads, transformer_units, mlp_head_units,
                             num_classes, drop_out, reg, l2_weight, demograp
    if reg:
        activation = None
    else:
        activation = 'sigmoid'
    inputs = Input(shape=input_shape)
    patch_size = input_shape[0] / num_patches
    if demographic:
        normalized_inputs = tfa.layers.InstanceNormalization(axis=-1, epsilo
                                                             beta_initialize
                                                             gamma_initializ
        demo = inputs[:, :12, -1]

    else:
        normalized_inputs = tfa.layers.InstanceNormalization(axis=-1, epsilo
                                                             beta_initialize
                                                             gamma_initializ

    # patches = Reshape((num_patches, -1))(normalized_inputs)
    patches = Patches(patch_size=patch_size)(normalized_inputs)
    encoded_patches = PatchEncoder(num_patches=num_patches, projection_dim=p
    for i in range(transformer_layers):
        x1 = encoded_patches # LayerNormalization(epsilon=1e-6)(encoded_patc
        attention_output = MultiHeadAttention(
            num_heads=num_heads, key_dim=projection_dim, dropout=drop_out, k
            bias_regularizer=L2(l2_weight))(x1, x1)
        x2 = Add()([attention_output, encoded_patches])
        x3 = LayerNormalization(epsilon=1e-6)(x2)
        x3 = mlp(x3, transformer_units, drop_out, l2_weight)  # i *
        encoded_patches = Add()([x3, x2])

    x = LayerNormalization(epsilon=1e-6)(encoded_patches)
```

```
    x = GlobalAveragePooling1D()(x)
    #x = Concatenate()([x, demo])
    features = mlp(x, mlp_head_units, 0.0, l2_weight)

    logits = Dense(num_classes, kernel_regularizer=L2(l2_weight), bias_regul
                   activation=activation)(features)

    return tf.keras.Model(inputs=inputs, outputs=logits)



def create_hybrid_transformer_model(input_shape):
    transformer_units =  [32,32]
    transformer_layers = 2
    num_heads = 4
    l2_weight = 0.001
    drop_out= 0.25
    mlp_head_units = [256, 128]
    num_patches=30
    projection_dim=  32

    # Conv1D(32...
    input1 = Input(shape=input_shape)
    conv11 = Conv1D(16, 256)(input1) #13
    conv12 = Conv1D(16, 256)(input1) #13
    conv13 = Conv1D(16, 256)(input1) #13

    pwconv1 = SeparableConvolution1D(32, 1)(input1)
    pwconv2 = SeparableConvolution1D(32, 1)(pwconv1)

    conv21 = Conv1D(16, 256)(conv11) # 7
    conv22 = Conv1D(16, 256)(conv12) # 7
    conv23 = Conv1D(16, 256)(conv13) # 7

    concat = keras.layers.concatenate([conv21, conv22, conv23], axis=-1)
    concat = Dense(64, activation=relu)(concat) #192
    concat = Dense(64, activation=sigmoid)(concat) #192
    concat = SeparableConvolution1D(32,1)(concat)
    concat = keras.layers.concatenate([concat, pwconv2], axis=1)

    ##########################################################################
    patch_size = input_shape[0] / num_patches

    normalized_inputs = tfa.layers.InstanceNormalization(axis=-1, epsilon=1e
                                                         beta_initialize
                                                         gamma_initializ

    # patches = Reshape((num_patches, -1))(normalized_inputs)
    patches = Patches(patch_size=patch_size)(normalized_inputs)
    encoded_patches = PatchEncoder(num_patches=num_patches, projection_dim=p
```

```
    for i in range(transformer_layers):
        x1 = encoded_patches # LayerNormalization(epsilon=1e-6)(encoded_patc
        attention_output = MultiHeadAttention(
            num_heads=num_heads, key_dim=projection_dim, dropout=drop_out, k
            bias_regularizer=L2(l2_weight))(x1, x1)
        x2 = Add()([attention_output, encoded_patches])
        x3 = LayerNormalization(epsilon=1e-6)(x2)
        x3 = mlp(x3, transformer_units, drop_out, l2_weight)  # i *
        encoded_patches = Add()([x3, x2])

    x = LayerNormalization(epsilon=1e-6)(encoded_patches)
    x = GlobalAveragePooling1D()(x)
    #x = Concatenate()([x, demo])
    features = mlp(x, mlp_head_units, 0.0, l2_weight)

    logits = Dense(1, kernel_regularizer=L2(l2_weight), bias_regularizer=L2(
                   activation='sigmoid')(features)

    ##########################################################################

    model = Model(inputs=input1, outputs=logits)
    return model
```

Supporting layers

There are several layers prior to, and after the transformer. As discussed above, these include several 1-D convolutions, activations, concatenations, normalizations, and fully-connected layers. Code for these layers is as follows:

In [ ]:
```
import keras
from keras import Input, Model
from keras.layers import Dense, Flatten, MaxPooling2D, Conv2D, BatchNormaliz
    Reshape, GRU, Conv1D, MaxPooling1D, Activation, Dropout, GlobalAveragePo
    LayerNormalization, SeparableConvolution1D
from keras.models import Sequential
from keras.activations import relu, sigmoid
from keras.regularizers import l2
import tensorflow_addons as tfa

def create_cnn_model(input_shape):
    model = Sequential()
    for i in range(5): # 10
        model.add(Conv1D(45, 32, padding='same'))
        model.add(BatchNormalization())
        model.add(Activation(relu))
        model.add(MaxPooling1D())
        model.add(Dropout(0.5))
```

```
    model.add(Flatten())
    for i in range(2): #4
        model.add(Dense(512))
        model.add(BatchNormalization())
        model.add(Activation(relu))
        model.add(Dropout(0.5))

    model.add(Dense(1, activation='sigmoid'))

    return model


def create_cnnlstm_model(input_a_shape, weight=1e-3):
    cnn_filters = 32 # 128
    cnn_kernel_size = 4 # 4
    input1 = Input(shape=input_a_shape)
    input1 = tfa.layers.InstanceNormalization(axis=-1, epsilon=1e-6, center=
                                              beta_initializer="glorot_unifo
                                              gamma_initializer="glorot_unif
    x1 = Conv1D(cnn_filters, cnn_kernel_size, activation='relu')(input1)
    x1 = Conv1D(cnn_filters, cnn_kernel_size, activation='relu')(x1)
    x1 = BatchNormalization()(x1)
    x1 = MaxPooling1D()(x1)

    x1 = Conv1D(cnn_filters, cnn_kernel_size, activation='relu')(x1)
    x1 = BatchNormalization()(x1)
    x1 = MaxPooling1D()(x1)

    x1 = Conv1D(cnn_filters, cnn_kernel_size, activation='relu')(x1)
    x1 = BatchNormalization()(x1)
    x1 = MaxPooling1D()(x1)

    x1 = LSTM(32, return_sequences=True)(x1) #256
    x1 = LSTM(32, return_sequences=True)(x1) #256
    x1 = LSTM(32)(x1) #256
    x1 = Flatten()(x1)

    x1 = Dense(32, activation='relu')(x1) #64
    x1 = Dense(32, activation='relu')(x1) #64
    outputs = Dense(1, activation='sigmoid')(x1)

    model = Model(inputs=input1, outputs=outputs)
    return model


def create_semscnn_model(input_a_shape):
    input1 = Input(shape=input_a_shape)
    # input1 = tfa.layers.InstanceNormalization(axis=-1, epsilon=1e-6, cente
    #                                           beta_initializer="glorot_uni
```

```
    #                                                gamma_initializer="glorot_ur
    x1 = Conv1D(45, 32, strides=1)(input1) #kernel_size=11
    x1 = Conv1D(45, 32, strides=2)(x1) #64 kernel_size=11
    x1 = BatchNormalization()(x1)
    x1 = Activation(relu)(x1)
    x1 = MaxPooling1D()(x1)

    x1 = Conv1D(45, 32, strides=2)(x1) #64 kernel_size=11
    x1 = BatchNormalization()(x1)
    x1 = Activation(relu)(x1)
    x1 = MaxPooling1D()(x1)

    x1 = Conv1D(45, 32, strides=2)(x1) #64 kernel_size=11
    x1 = BatchNormalization()(x1)
    x1 = Activation(relu)(x1)
    x1 = MaxPooling1D()(x1)

    squeeze = Flatten()(x1)
    excitation = Dense(128, activation='relu')(squeeze)
    excitation = Dense(64, activation='relu')(excitation)
    logits = Dense(1, activation='sigmoid')(excitation)
    model = Model(inputs=input1, outputs=logits)
    return model


model_dict = {

    "cnn": create_cnn_model((60 * 32, 3)),
    "sem-mscnn": create_semscnn_model((60 * 32, 3)),
    "cnn-lstm": create_cnnlstm_model((60 * 32, 3)),
    "hybrid": create_hybrid_transformer_model((60 * 32, 3)),
}


def get_model(config):
    if config["model_name"].split('_')[0] == "Transformer":
        return create_transformer_model(input_shape=(60 * 32, len(config["ch
                                        num_patches=config["num_patches"], p
                                        transformer_layers=config["transform
                                        transformer_units=[config["transform
                                                           config["transform
                                        mlp_head_units=[256, 128], num_class
                                        reg=config["regression"], l2_weight=
    else:
        return model_dict.get(config["model_name"].split('_')[0])

def run_model():
    config = {
        "model_name": "hybrid",
        "regression": False,
```

```
        "transformer_layers": 4,  # best 5
        "drop_out_rate": 0.25,
        "num_patches": 20,  # best
        "transformer_units": 32,  # best 32
        "regularization_weight": 0.001,  # best 0.001
        "num_heads": 4,
        "epochs": 100,  # best
        "channels": [14, 18, 19, 20],
    }
    model = get_model(config)
    model.build(input_shape=(1, 60 * 32, 10))
    print(model.summary())
    return model
```

Model training

From data preprocessing and model architecture, we can train the model. As discussed previously, we do not provide raw data for licensing reasons. Instead, we provide preprocessed data on a limited basis, and also provide pre-trained model weights. To minimize resource requirements and runtime when running this notebook in this section, we will show training on a limited subset of the preprocessed data and limited number of epochs. We will also take steps to minimize resource requirements and runtime in the next section.

Supporting code

First, some utility code to allow us to manipulate training data as necessary during the subsequent model training loop.

In [ ]:
```
import numpy as np
import numpy.typing as npt

def _replace_final_dim(orig_x: npt.NDArray, final_dim_size: int) -> npt.NDAr
    '''
    Given an NDArray `orig_x`, return a new NDArray of the same shape,
    except with final dimension `final_dim_size`.
    '''
    orig_x_shape = orig_x.shape
    x_transform = np.zeros(
        (*orig_x_shape[:-1], final_dim_size)
    )
    assert x_transform.shape[:-1] == orig_x.shape[:-1]
```

```
    return x_transform

def transform_for_channels(x: npt.NDArray, channels: list[int]) -> npt.NDArr
    '''
    Returns an NDArray whose shape is identical to that of the `x` parameter
    except the final dimension is of size `len(channels)`

    Generally speaking, x has a shape similar to
    (NUM_FOLDS, 530, 1920, NUM_CHANNELS). In this array, each fold thus
    has shape (530, 1920, NUM_CHANNELS).

    Since we're trying to only extract `num_channels` out into the last
    dimension, we need to create an NDArray whose final dimension matches
    that number of channels.

    The value returned from this function, which we'll call `x_transform`,
    is compatible with this number of channels since its last dimension is
    the number of channels we're trying to extract.
    '''

    num_channels = len(channels)
    assert (len(x.shape)) == 4
    x_transform = _replace_final_dim(
        orig_x=x,
        final_dim_size=num_channels,
    )
    return x_transform

def concat_all_folds(orig: npt.NDArray, except_fold: int) -> npt.NDArray:
    assert(except_fold) >= 0
    assert len(orig.shape) > 0
    # how to initialize our concatenated array:
    #
    # - if except_fold is 0, initialize with index 1
    # - if except_fold is 1, initialize with index 0
    #
    # in either case, we want to start at index 2 since we skip one of the
    # previous indices (0 and 1) and choose to start with the other.
    concat = orig[1] if except_fold == 0 else orig[0]
    for i in range(2, orig.shape[0]):
        if i == except_fold:
            continue
        concat = np.concatenate((concat, orig[i]))
    return concat
```

Training loop

Next, the actual training loop.

In [ ]:
```
import os
from typing import Any
import keras
import keras.metrics
import numpy as np
from keras.callbacks import LearningRateScheduler, EarlyStopping
from keras.losses import BinaryCrossentropy
from sklearn.utils import shuffle
import numpy.typing as npt

# from models.models import get_model
# from channels import transform_for_channels
# from folds import concat_all_folds

THRESHOLD = 1
FOLD = 5

def lr_schedule(epoch, lr):
    if epoch > 50 and (epoch - 1) % 5 == 0:
        lr *= 0.5
    return lr


def train(
        config,
        fold: int | None = None,
        force_retrain: bool = False
):
    print(f'training with config {config}, fold={fold}')

    data = np.load(config["data_path"], allow_pickle=True)

    x, y_apnea, y_hypopnea = data['x'], data['y_apnea'], data['y_hypopnea']
    print(
        f'x={x.shape}, y_apnea={y_apnea.shape}, y_hypopnea={y_hypopnea.shape
    )
    y = y_apnea + y_hypopnea
    ##########################################################################
    # Channel selection

    chans = config["channels"]
    x_transform = transform_for_channels(x=x, channels=chans)
    print(f'Extracting channels {chans}')
    max_fold = min(FOLD, x_transform.shape[0])
    if max_fold < x_transform.shape[0]:
        print(
            f'WARNING: only looking at the first {max_fold} of '
            f'{x_transform.shape[0]} total folds in X'
        )
```

```
    # for i in range(FOLD):
    for i in range(max_fold):
        x_transform[i], y[i] = shuffle(x_transform[i], y[i])
        x[i] = np.nan_to_num(x[i], nan=-1)
        if config["regression"]:
            y[i] = np.sqrt(y[i])
            y[i][y[i] != 0] += 2
        else:
            y[i] = np.where(y[i] >= THRESHOLD, 1, 0)

        replace = x[i][:, :, chans]

        x_transform[i] = replace  # CHANNEL SELECTION

    ###############################################################################
    #
    # The original code for this is taken from the following link:
    #
    # https://github.com/healthylaife/Pediatric-Apnea-Detection/blob/6dc5ec8
    #
    # I (Aaron) think that in the inner loop, they're just trying to create
    # one big NDArray with the concatenation of all the folds except for the
    # one on which they're currently on in the outer loop.
    #
    # Then, they train on the concatenated array. In other words, the outer
    # loop behaves similarly to epochs, with a small twist.
    #
    # They used to have the logic to do this inside the outer loop,
    # but I pulled it out.
    #
    # also note, the folds selection (commented below) didn't work because
    # they pass fold=0 into this function, which results in no training
    # whatsoever.
    folds = range(max_fold)
    # folds = range(FOLD) if fold is None else range(fold)
    print(f'iterating over {folds} fold(s)')
    for fold in folds:
        base_model_path = config["model_path"]
        model_path = f"{base_model_path}/{str(fold)}"
        if (
            os.path.exists(model_path) and
            not force_retrain
        ):
            print(
                f'Training fold {fold}: force_retrain==False and '
                f'{model_path} already exists, skipping.'
            )
            continue
        x_train = concat_all_folds(orig=x_transform, except_fold=fold)
        y_train = concat_all_folds(orig=y, except_fold=fold)
```

```
        model = get_model(config)
        if config["regression"]:
            model.compile(optimizer="adam", loss=BinaryCrossentropy())
            early_stopper = EarlyStopping(monitor='val_loss', patience=10, r

        else:
            model.compile(optimizer="adam", loss=BinaryCrossentropy(),
                          metrics=[keras.metrics.Precision(), keras.metrics.
            early_stopper = EarlyStopping(monitor='val_loss', patience=10, r
        lr_scheduler = LearningRateScheduler(lr_schedule)
        model.fit(x=x_train, y=y_train, batch_size=512, epochs=config["epoch
                  callbacks=[early_stopper, lr_scheduler])
        ##########################################################################
        print(f"saving model for fold {fold} to {model_path}")
        model.save(model_path)
        keras.backend.clear_session()


def train_age_seperated(config):
    data = np.load(config["data_path"], allow_pickle=True)
    x, y_apnea, y_hypopnea = data['x'], data['y_apnea'], data['y_hypopnea']
    y = y_apnea + y_hypopnea
    ##########################################################################
    for i in range(10):
        x[i], y[i] = shuffle(x[i], y[i])
        x[i] = np.nan_to_num(x[i], nan=-1)
        if config["regression"]:
            y[i] = np.sqrt(y[i])
            y[i][y[i] != 0] += 2
        else:
            y[i] = np.where(y[i] >= THRESHOLD, 1, 0)

        x[i] = x[i][:, :, config["channels"]]  # CHANNEL SELECTION

    ##########################################################################
    first = True
    for i in range(10):
        if first:
            x_train = x[i]
            y_train = y[i]
            first = False
        else:
            x_train = np.concatenate((x_train, x[i]))
            y_train = np.concatenate((y_train, y[i]))

    model = get_model(config)
    if config["regression"]:
        model.compile(optimizer="adam", loss=BinaryCrossentropy())
        early_stopper = EarlyStopping(
```

```
            monitor='val_loss',
            patience=10,
            restore_best_weights=True
        )

    else:
        model.compile(optimizer="adam", loss=BinaryCrossentropy(),
                      metrics=[keras.metrics.Precision(), keras.metrics.Reca
        early_stopper = EarlyStopping(
            monitor='val_loss',
            patience=10,
            restore_best_weights=True
        )
    lr_scheduler = LearningRateScheduler(lr_schedule)
    model.fit(
        x=x_train,
        y=y_train,
        batch_size=512,
        epochs=config["epochs"],
        validation_split=0.1,
        callbacks=[early_stopper, lr_scheduler]
    )
    ##########################################################################
    model.save(config["model_path"] + str(0))
    keras.backend.clear_session()

# the original repository indicated that the best value for this constant
# is 200. we are setting it to a much lower number so we can illustrate
# training working without requiring a large amount of computing power
# or running for an excessively long time.
#
# feel free to increase this value, but be mindful that if you do, we make
# no guarantees about required resources or expected runtime.
NUM_EPOCHS=2
def build_configs() -> dict[str, dict[str, Any]]:
    sig_dict = {"EOG": [0, 1],
                "EEG": [2, 3],
                "RESP": [5, 6],
                "SPO2": [9],
                "CO2": [10],
                "ECG": [11, 12],
                "DEMO": [13],
                }

    channel_list = [
        ["ECG", "SPO2"],
    ]
    configs: dict[str, dict[str, Any]] = {}
    for ch in channel_list:
            chs = []
```

```
            chstr = ""
            for name in ch:
                chstr += name
                chs = chs + sig_dict[name]
            if chstr in configs:
                raise ValueError(
                    f"tried to create a config for channel {chstr}, "
                    "but one already existed"
                )
            configs[chstr] = {
                "data_path": PREPROCESS_OUT_PATH,
                "model_path": MODEL_OUT_PATH,
                "model_name": "sem-mscnn_" + chstr,
                "regression": False,

                "transformer_layers": 5,  # best 5
                "drop_out_rate": 0.25,  # best 0.25
                "num_patches": 30,  # best 30 TBD
                "transformer_units": 32,  # best 32
                "regularization_weight": 0.001,  # best 0.001
                "num_heads": 4,
                "epochs": NUM_EPOCHS,
                "channels": chs,
            }
    return configs

def run_training():

    configs = build_configs()

    for chstr, config in configs.items():
        print(
            f"----------\n"
            f"training channel {chstr}..."
        )
        train(config=config, fold=0, force_retrain=True)
        print("done.")

if SKIP_TRAINING:
    print("SKIP_TRAINING is set to True, so not running training loop")
else:
    print('running training...')
    run_training()
    print("done.")
```

```
running training...
----------
training channel ECGSPO2...
training with config {'data_path': '/root/data/physionet.org/nch_30x64.npz',
'model_path': 'original/weights/semscnn_ecgspo2/f', 'model_name': 'sem-mscnn
_ECGSPO2', 'regression': False, 'transformer_layers': 5, 'drop_out_rate': 0.
25, 'num_patches': 30, 'transformer_units': 32, 'regularization_weight': 0.0
01, 'num_heads': 4, 'epochs': 2, 'channels': [11, 12, 9]}, fold=0
x=(5, 1836, 1920, 14), y_apnea=(5, 1836), y_hypopnea=(5, 1836)
Extracting channels [11, 12, 9]
iterating over range(0, 5) fold(s)
Epoch 1/2
13/13 [==============================] - 24s 2s/step - loss: 0.5972 - precis
ion: 0.2472 - recall: 0.0121 - val_loss: 9.5304 - val_precision: 0.0000e+00
- val_recall: 0.0000e+00 - lr: 0.0010
Epoch 2/2
13/13 [==============================] - 21s 2s/step - loss: 0.5907 - precis
ion: 0.0000e+00 - recall: 0.0000e+00 - val_loss: 5.6446 - val_precision: 0.0
000e+00 - val_recall: 0.0000e+00 - lr: 0.0010
saving model for fold 0 to original/weights/semscnn_ecgspo2/f/0
INFO:tensorflow:Assets written to: original/weights/semscnn_ecgspo2/f/0/asse
ts
INFO:tensorflow:Assets written to: original/weights/semscnn_ecgspo2/f/0/asse
ts
Epoch 1/2
13/13 [==============================] - 24s 2s/step - loss: 0.5884 - precis
ion: 0.2975 - recall: 0.0886 - val_loss: 13.9062 - val_precision: 0.0000e+00
- val_recall: 0.0000e+00 - lr: 0.0010
Epoch 2/2
13/13 [==============================] - 21s 2s/step - loss: 0.5776 - precis
ion: 0.0000e+00 - recall: 0.0000e+00 - val_loss: 10.6658 - val_precision: 0.
0000e+00 - val_recall: 0.0000e+00 - lr: 0.0010
saving model for fold 1 to original/weights/semscnn_ecgspo2/f/1
INFO:tensorflow:Assets written to: original/weights/semscnn_ecgspo2/f/1/asse
ts
INFO:tensorflow:Assets written to: original/weights/semscnn_ecgspo2/f/1/asse
ts
Epoch 1/2
10/10 [==============================] - 18s 2s/step - loss: 0.6213 - precis
ion: 0.3626 - recall: 0.0222 - val_loss: 21.3808 - val_precision: 0.0000e+00
- val_recall: 0.0000e+00 - lr: 0.0010
Epoch 2/2
10/10 [==============================] - 16s 2s/step - loss: 0.6013 - precis
ion: 0.5000 - recall: 0.0054 - val_loss: 16.4141 - val_precision: 0.0000e+00
- val_recall: 0.0000e+00 - lr: 0.0010
saving model for fold 2 to original/weights/semscnn_ecgspo2/f/2
INFO:tensorflow:Assets written to: original/weights/semscnn_ecgspo2/f/2/asse
ts
```

```
INFO:tensorflow:Assets written to: original/weights/semscnn_ecgspo2/f/2/asse
ts
Epoch 1/2
10/10 [==============================] - 18s 2s/step - loss: 0.5861 - precis
ion: 0.3000 - recall: 0.0022 - val_loss: 5.0613 - val_precision: 0.0000e+00
- val_recall: 0.0000e+00 - lr: 0.0010
Epoch 2/2
10/10 [==============================] - 17s 2s/step - loss: 0.5725 - precis
ion: 0.6667 - recall: 0.0015 - val_loss: 4.7401 - val_precision: 0.0000e+00
- val_recall: 0.0000e+00 - lr: 0.0010
saving model for fold 3 to original/weights/semscnn_ecgspo2/f/3
INFO:tensorflow:Assets written to: original/weights/semscnn_ecgspo2/f/3/asse
ts
INFO:tensorflow:Assets written to: original/weights/semscnn_ecgspo2/f/3/asse
ts
Epoch 1/2
10/10 [==============================] - 19s 2s/step - loss: 0.5232 - precis
ion: 0.0000e+00 - recall: 0.0000e+00 - val_loss: 0.6198 - val_precision: 0.0
000e+00 - val_recall: 0.0000e+00 - lr: 0.0010
Epoch 2/2
10/10 [==============================] - 15s 1s/step - loss: 0.5158 - precis
ion: 0.0000e+00 - recall: 0.0000e+00 - val_loss: 0.5871 - val_precision: 0.0
000e+00 - val_recall: 0.0000e+00 - lr: 0.0010
saving model for fold 4 to original/weights/semscnn_ecgspo2/f/4
INFO:tensorflow:Assets written to: original/weights/semscnn_ecgspo2/f/4/asse
ts
INFO:tensorflow:Assets written to: original/weights/semscnn_ecgspo2/f/4/asse
ts
done.
done.
```

Results and evaluations

In a traditional machine learning pipeline, we would train a model and then immediately evaluate it on a test data set. As with the last section, however, we take steps to minimize resource requirements and runtime. To that end, we have run training on the full dataset and saved the resulting model's weights (which you can find in our repository's `original/weights` directory).

In this section, however, we've built the option to instantiate our model from those weights, then evaluate it. The results of this evaluation determine how well we were able to reproduce the results claimed in the original paper.

The code shown in this section tests our preloaded model. As with the previous section, testing and evaluation code is extensive, so we split it into two separate sections.

Metrics reporting and calculation

Below, we have utility code for metrics collection and reporting. This code will be used by subsequent evaluation code.

In [ ]:
```
import os
import tensorflow as tf
import tensorflow_addons as tfa
import numpy as np
from sklearn.metrics import confusion_matrix, f1_score, average_precision_sc


class FromLogitsMixin:
    def __init__(self, from_logits=False, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.from_logits = from_logits

    def update_state(self, y_true, y_pred, sample_weight=None):
        if self.from_logits:
            y_pred = tf.nn.sigmoid(y_pred)
        return super().update_state(y_true, y_pred, sample_weight)


class AUC(FromLogitsMixin, tf.metrics.AUC):
    ...


class BinaryAccuracy(FromLogitsMixin, tf.metrics.BinaryAccuracy):
    ...


class TruePositives(FromLogitsMixin, tf.metrics.TruePositives):
    ...


class FalsePositives(FromLogitsMixin, tf.metrics.FalsePositives):
    ...


class TrueNegatives(FromLogitsMixin, tf.metrics.TrueNegatives):
    ...


class FalseNegatives(FromLogitsMixin, tf.metrics.FalseNegatives):
    ...
```

```
class Precision(FromLogitsMixin, tf.metrics.Precision):
    ...


class Recall(FromLogitsMixin, tf.metrics.Recall):
    ...


class F1Score(FromLogitsMixin, tfa.metrics.F1Score):
    ...


class Result:
    def __init__(self):
        self.accuracy_list = []
        self.sensitivity_list = []
        self.specificity_list = []
        self.f1_list = []
        self.auroc_list = []
        self.auprc_list = []
        self.precision_list = []

    def add(self, y_test, y_predict, y_score):
        C = confusion_matrix(y_test, y_predict, labels=(1, 0))
        TP, TN, FP, FN = C[0, 0], C[1, 1], C[1, 0], C[0, 1]

        acc, sn, sp, pr = 1. * (TP + TN) / (TP + TN + FP + FN), 1. * TP / (T
                TP + FP)
        acc = 1. * (TP + TN) / (TP + TN + FP + FN)
        sn = 1. * TP / (TP + FN)
        sp = 1. * TN / (TN + FP)
        pr = 1. * TP / (TP + FP) if TP + FP != 0 else 0 # define precision t
        f1 = f1_score(y_test, y_predict)
        auc = roc_auc_score(y_test, y_score)
        auprc = average_precision_score(y_test, y_score)

        self.accuracy_list.append(acc * 100)
        self.precision_list.append(pr * 100)
        self.sensitivity_list.append(sn * 100)
        self.specificity_list.append(sp * 100)
        self.f1_list.append(f1 * 100)
        self.auroc_list.append(auc * 100)
        self.auprc_list.append(auprc * 100)


    def get(self):
        out_str = "=========================================================
        out_str += str(self.accuracy_list) + " \n"
        out_str += str(self.precision_list) + " \n"
        out_str += str(self.sensitivity_list) + " \n"
```

```
        out_str += str(self.specificity_list) + " \n"
        out_str += str(self.f1_list) + " \n"
        out_str += str(self.auroc_list) + " \n"
        out_str += str(self.auprc_list) + " \n"
        out_str += str("Accuracy: %.2f -+ %.3f" % (np.mean(self.accuracy_lis
        out_str += str("Precision: %.2f -+ %.3f" % (np.mean(self.precision_l
        out_str += str(
            "Recall: %.2f -+ %.3f" % (np.mean(self.sensitivity_list), np.std
        out_str += str(
            "Specifity: %.2f -+ %.3f" % (np.mean(self.specificity_list), np.
        out_str += str("F1: %.2f -+ %.3f" % (np.mean(self.f1_list), np.std(s
        out_str += str("AUROC: %.2f -+ %.3f" % (np.mean(self.auroc_list), np
        out_str += str("AUPRC: %.2f -+ %.3f" % (np.mean(self.auprc_list), np

        out_str += str("$ %.1f \pm %.1f$" % (np.mean(self.accuracy_list), np
        out_str += str("$%.1f \pm %.1f$" % (np.mean(self.precision_list), np
        out_str += str("$%.1f \pm %.1f$" % (np.mean(self.sensitivity_list),
        out_str += str("$%.1f \pm %.1f$" % (np.mean(self.f1_list), np.std(se
        out_str += str("$%.1f \pm %.1f$" % (np.mean(self.auroc_list), np.std

        return out_str

    def print(self):
        print(self.get())

    def save(self, path, config):
        enclosing_dir = os.path.dirname(path)
        if not os.path.exists(enclosing_dir):
            print(
                f'directory {enclosing_dir} did not exist, creating it for '
                'result saving'
            )
            os.makedirs(enclosing_dir)

        file = open(path, "w+")
        file.write(str(config))
        file.write("\n")
        file.write(self.get())
        file.flush()
        file.close()
```

Using pretrained model

The `TEST_FROM_CHECKPOINT` flag indicates whether we should test from a model loaded from pretrained weights (`True`), or the model we just tested (`False`). If that flag is set to `True` , we expect a zip archive of the weights to be downloadable from the value in `CHECKPOINT_URL` . The code below checks for the flag and downloads weights

if necessary.

In [ ]:
```
import os
from urllib.request import urlretrieve
import zipfile


if TEST_FROM_CHECKPOINT:
    print(
        f'TEST_FROM_CHECKPOINT is True, downloading {CHECKPOINT_URL} '
        f'and saving to {MODEL_OUT_PATH}'
    )
    zip_path = "./model.zip"
    # first download the zip to the current working dir
    urlretrieve(CHECKPOINT_URL, zip_path)
    if not os.path.exists(MODEL_OUT_PATH):
        print(f'{MODEL_OUT_PATH} did not exist, creating')
        os.makedirs(MODEL_OUT_PATH)
    with zipfile.ZipFile(zip_path, 'r') as zip_ref:
        zip_ref.extractall('original')
    os.remove(zip_path)

else:
    if not os.path.exists(MODEL_OUT_PATH):
        raise FileNotFoundError(
            f"TEST_FROM_CHECKPOINT is False, but {MODEL_OUT_PATH} was not "
            f"found. please make sure model training succeeded"
        )
```

```
TEST_FROM_CHECKPOINT is True, downloading https://dlhproject.sfo3.cdn.digita
loceanspaces.com/weights-2024-14-14.zip and saving to original/weights/semsc
nn_ecgspo2/f
```

Metrics calculation

The below code evaluates the model and reports it using the above reporting code.

In [ ]:
```
import os
from datetime import datetime

import numpy as np
from sklearn.utils import shuffle
import tensorflow as tf

def sigmoid(x):
    return 1 / (1 + np.exp(-x))
```

```
THRESHOLD = 1
FOLD = 5

def add_noise_to_data(a):
    """
    Stand-in function for adding noise to data.

    This function is used in the codebase, hidden behind a
    config flag, and not defined anywhere in the original research
    repository, so we have defined it here as the identity function,
    just so things run properly.
    """
    print("WARNING: 'test_noise_snr' config option is a no-op")
    return a


def test(config: dict[str, str], fold=None):
    data_path = config['data_path']
    print(f'testing from {data_path}')
    data = np.load(data_path, allow_pickle=True)
    ###########################################################################
    x, y_apnea, y_hypopnea = data["x"], data["y_apnea"], data["y_hypopnea"]
    x_transform = transform_for_channels(x=x, channels=config["channels"])
    y = y_apnea + y_hypopnea

    max_fold = min(FOLD, x_transform.shape[0])
    if max_fold < x_transform.shape[0]:
        print(
            f'WARNING: only looking at the first {max_fold} of '
            f'{x_transform.shape[0]} total folds in X'
        )

    # for i in range(FOLD):
    for i in range(max_fold):
        x_transform[i], y[i] = shuffle(x_transform[i], y[i])
        x[i] = np.nan_to_num(x[i], nan=-1)
        y[i] = np.where(y[i] >= THRESHOLD, 1, 0)
        x_transform[i] = x[i][:, :, config["channels"]]
    ###########################################################################
    result = Result()
    # folds = range(FOLD) if fold is None else [fold]
    folds = range(max_fold)
    for fold in folds:
        base_model_path = config["model_path"]
        model_path = f"{base_model_path}/{str(fold)}"
        if not os.path.exists(model_path):
            print(
                f"WARNING: model path {model_path} does not exist, skipping!
            )
            continue
```

```
        x_test = x_transform[fold]
        # NOTE: this config key is not set in both `main_chat.py` and
        # `main_nch.py`. if it were, the code under this `if` would fail
        # because there is no `add_noise_to_data` function in this repositor
        if config.get("test_noise_snr"):
            x_test = add_noise_to_data(x_test, config["test_noise_snr"])

        y_test = y[
            fold
        ]  # For MultiClass keras.utils.to_categorical(y[fold], num_classes=
        model = tf.keras.models.load_model(model_path, compile=False)

        predict = model.predict(x_test)
        y_score = predict
        y_predict = np.where(
            predict > 0.5, 1, 0
        )  # For MultiClass np.argmax(y_score, axis=-1)

        result.add(y_test, y_predict, y_score)

    print(
        '\n----------\n'
        'results:\n'
    )
    result.print()
    model_name = config["model_name"]
    timestamp = datetime.now().strftime("%Y%m%d-%H%M") # Format the date and
    results_file = os.path.join('results', f"{model_name}-{timestamp}.txt")
    print(
        f'done, saving to {results_file}\n'
        '----------\n'
    )

    result.save(path=results_file, config=config)

    del data, x_test, y_test, model, predict, y_score, y_predict


def test_age_seperated(config):
    x = []
    y_apnea = []
    y_hypopnea = []
    for i in range(10):
        data = np.load(config["data_path"] + str(i) + ".npz", allow_pickle=T
        x.append(data["x"])
        y_apnea.append(data["y_apnea"])
        y_hypopnea.append(data["y_hypopnea"])
    ##########################################################################
    y = np.array(y_apnea) + np.array(y_hypopnea)
    for i in range(10):
```

```
        x[i], y[i] = shuffle(x[i], y[i])
        x[i] = np.nan_to_num(x[i], nan=-1)
        y[i] = np.where(y[i] >= THRESHOLD, 1, 0)
        x[i] = x[i][:, :, config["channels"]]
    ##########################################################################
    result = Result()

    for fold in range(10):
        x_test = x[fold]
        if config.get("test_noise_snr"):
            x_test = add_noise_to_data(x_test, config["test_noise_snr"])

        y_test = y[
            fold
        ]  # For MultiClass keras.utils.to_categorical(y[fold], num_classes=

        model = tf.keras.models.load_model(config["model_path"] + str(0), cc

        predict = model.predict(x_test)
        y_score = predict
        y_predict = np.where(
            predict > 0.5, 1, 0
        )  # For MultiClass np.argmax(y_score, axis=-1)

        result.add(y_test, y_predict, y_score)

    result.print()
    save_file = os.path.join('results', config["model_name"], ".txt")
    result.save(save_file, config)
    print(f"results saved to {save_file}")


    del data, x_test, y_test, model, predict, y_score, y_predict

configs = build_configs()
for chstr, config in configs.items():

    print(f'testing on channel {chstr}')
    test(config)
```

```
testing on channel ECGSPO2
testing from /root/data/physionet.org/nch_30x64.npz
58/58 [==============================] - 2s 23ms/step
58/58 [==============================] - 1s 21ms/step
/tmp/ipykernel_250794/2644645364.py:69: RuntimeWarning: invalid value encoun
tered in scalar divide
  acc, sn, sp, pr = 1. * (TP + TN) / (TP + TN + FP + FN), 1. * TP / (TP + F
N), 1. * TN / (TN + FP), 1. * TP / (
```

```
58/58 [==============================] - 1s 22ms/step
58/58 [==============================] - 1s 22ms/step
58/58 [==============================] - 1s 22ms/step

----------
results:

================================================================================
[81.97167755991286, 78.54030501089323, 80.3921568627451, 72.65795206971679,
49.673202614379086]
[0.0, 0, 25.0, 0.0, 75.0]
[0.0, 0.0, 0.27932960893854747, 0.0, 0.646551724137931]
[99.86728599867286, 100.0, 99.79702300405954, 99.8502994011976, 99.779735682
81938]
[0.0, 0.0, 0.5524861878453038, 0.0, 1.282051282051282]
[52.41396280377488, 49.5775748572555, 50.043846054989004, 51.6252994011976,
50.32321652362145]
[18.90981372113327, 20.806383980449116, 19.82803346339212, 27.73724284468449
5, 51.24596446326738]
Accuracy: 72.65 -+ 11.912
Precision: 20.00 -+ 29.155
Recall: 0.19 -+ 0.255
Specifity: 99.86 -+ 0.078
F1: 0.37 -+ 0.505
AUROC: 50.80 -+ 1.056
AUPRC: 27.71 -+ 12.175
$ 72.6 \pm 11.9$& $20.0 \pm 29.2$& $0.2 \pm 0.3$& $0.4 \pm 0.5$& $50.8 \pm
1.1$&
done, saving to results/sem-mscnn_ECGSPO2-20240414-2301.txt
----------
```

Model comparison

In [ ]:
```
# compare you model with others
# you don't need to re-run all other experiments, instead, you can directly
```

Discussion

Since we were unable to acquire all the data used in the paper, it's difficult to determine whether the paper is reproducible. The results we have with our current dataset are positive, however, so we believe the paper is reproducible.

Issues with paper reproduction

Given we encountered many significant issues during our paper reproduction efforts to date, we believe this paper is very difficult to reproduce. These issues have been discussed previously, but are summarized as follows:

- Data that are either impossible or very difficult, practically speaking, to acquire in a reasonable amount of time
- Open source code that is either very low quality or simply does not work
- Open source code that does not match exactly the architecture discussed in the paper

Suggestions for improvement

We estimate that data acquisition problems are outside the authors' control, so we focus our suggestions for improvement on code quality. Below are the highest-impact tasks we would suggest to the authors:

- Add clear documentation to each function, including descriptions of parameters and return values
- Add Python type hints to at least function parameters and return values
- Add complete information about the expected runtime environment, including required Python version(s) and dependency versions (including transitive dependencies)
- Do an audit to ensure that model architecture and evaluations match those described in the paper
- Add at least minimal testing to ensure all code runs successfully. Ideas for tests include:
  - Construct the model, load it from pretrained weights, and do several inferences, just to make sure it runs and can properly run inferences
  - Train the model for a small number of epochs to make sure loss is established and begins decreasing
  - Preprocess and load one sleep study to ensure preprocessing code runs properly

Plans for improvement

Between the draft due date and the final project due date, we plan to improve this notebook along several axes:

- Include more evaluations and visual aids to illustrate them (i.e. charts, graphs, etc...)

- Run at least one of the ablation studies from the paper and determine whether we reach the same conclusion as in the paper

References

1. Fayyaz H, Strang A, Beheshti R. Bringing At-home Pediatric Sleep Apnea Testing Closer to Reality: A Multi-modal Transformer Approach. Proc Mach Learn Res. 2023;219:167-185.