**Deep Learning: CPSC 8430**

**Homework 2**

**-Sandeep Varma Sangaraju**

**Github Link:**[**https://github.com/sangaraju-01/HW2\_Deep-learing**](https://github.com/sangaraju-01/HW2_Deep-learing)

**Outline of the Video caption generation**:

Computer vision and natural language processing are two related areas. I have encountered difficulties in generating descriptive captions in human language for videos using automated methods. It will be challenging to understand the dynamics of real-time videos. The technology we use should therefore enable variable-length input and output and account for temporal structure. The MSVD dataset is employed to evaluate the model, which utilizes an encoder-decoder framework. The encoding and decoding processes are performed using two Long Short-Term Memory (LSTM) units, with a deep neural network encoder used to learn video representation. The decoder then uses the learned representation to generate a sentence. To generate interesting captions, the Beam search algorithm is utilized.

**Introduction:**

It is difficult to implement and generate descriptive captions in human languages for videos, and also computer vision and natural language processing face difficulties for that. We need a model that can adapt to new images and generate text descriptions of different lengths based on input frames. To achieve this, we will develop a sequence-to-sequence. The model incorporates Long Short-Term Memory, a type of recurrent neural network, to improve its performance. RNNs, which can simulate sequential dynamics, have been successful in visual interpretation. The model aims to produce a word sequence output of varying lengths for image descriptions. For video descriptions, it needs to handle input word sequences and generate output picture sequences of different lengths.

The effectiveness of LSTMs in sequence-to-sequence tasks like speech recognition and machine translation has been demonstrated. To encode each frame individually during the encoding process, a stacked LSTM is utilized. A convolutional neural network that takes the intensity values of each frame as input is employed to provide input to the LSTM. The model generates a sentence by iterating through each frame and producing one word at a time. To generate captions for videos, LSTMs are used to combine the representation of each frame. The features of the CNN output are averaged using mean pools to obtain a single feature vector. LSTMs are used to merge the representation of all frames in one technique for generating video subtitles. However, the major disadvantage of this approach is that it ignores all temporal information and does not consider the order in which the video frames are presented.

Our model must be dynamic and capable of picking up new image data. Therefore, our objective is to build a seq-to-seq model to take frames as input and text output of varying lengths. To achieve this goal, we will use Long Short-Term Memory (LSTM), a type of Recurrent Neural Network (RNN), which is expected to enhance the performance of our model.

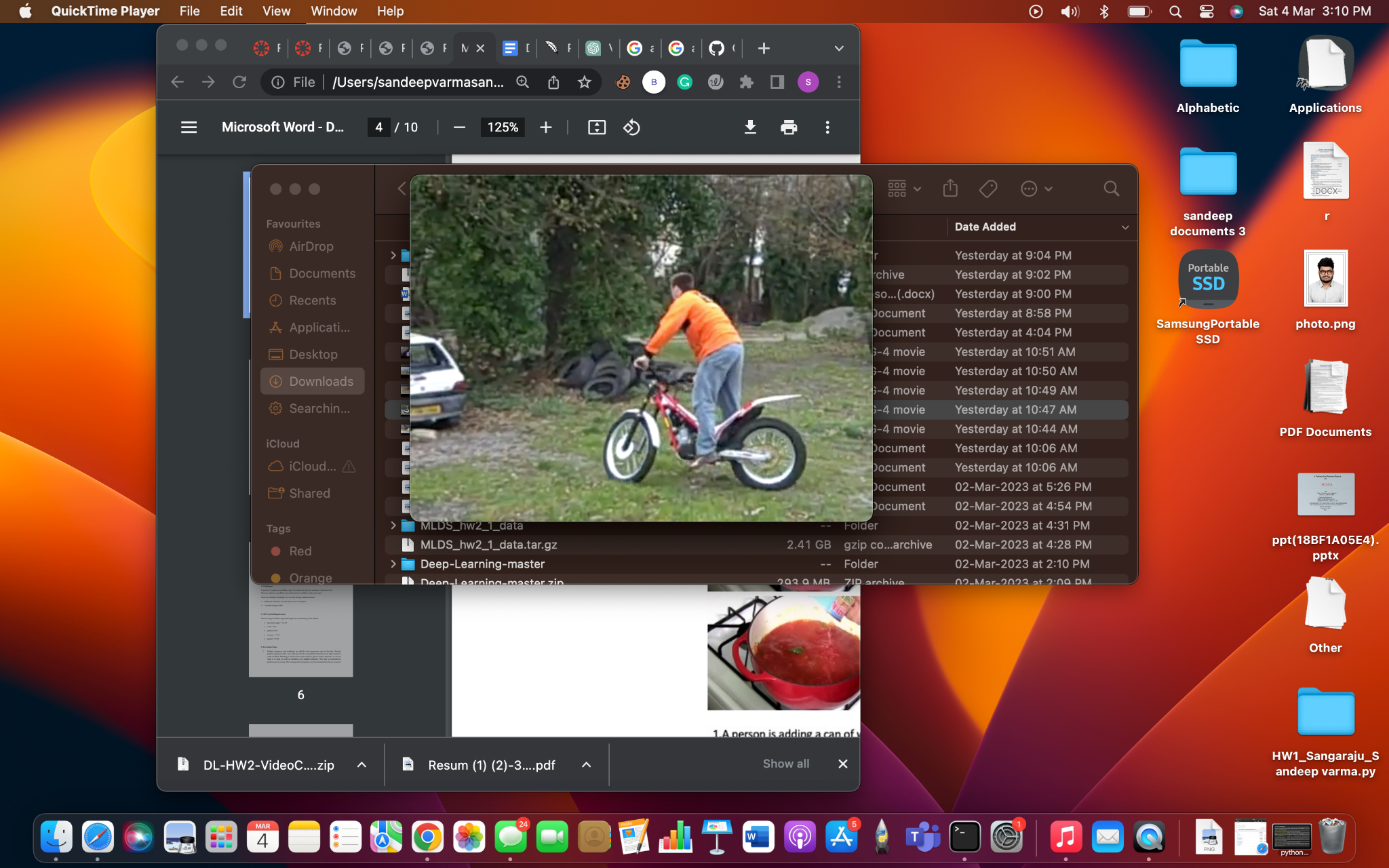
Due to its ability to dynamically emphasize important characteristics when necessary, attention mechanisms in neural networks have gained popularity. In many different tasks, including machine translation, picture captioning, and video captioning, attention models are helpful. At our model's decoder step, we've included an attention layer to help the model extract useful information from previous word inputs.

**Problem Statement:**

The Given problem is to create a Video Caption Generation for that Video content can be processed using a sequential-to-sequential approach. The input is a video, and the output is a sequence of captions describing the actions taking place in the video. Recurrent neural networks are a model that can perform this task by processing the input sequence and generating corresponding output captions.

**Dataset and Features:**

For this Video Caption Generation, I have used MSVDd(Microsoft Video Description dataset). In this dataset, we have taken 1450 videos for Training and 100 Videos for Testing. In this MSVD dataset, there are almost 1,20,000 Sentences. Employees at Mechanical Truck were Paid for visually scanning a brief video clip and then summarizing the event in one sentence. To optimize the advantages, we didn't remove any frames from the video while training. Instead, we preprocessed each video using pre-trained CNN VGG19 and saved the resulting features in the 80×4096 format.



A frame from the Training Dataset video(EPXsiQw9vvo\_1\_12).

1. A man is riding a bike.
2. A man riding a bike in the backyard.
3. A man doing Stunts with his bike.

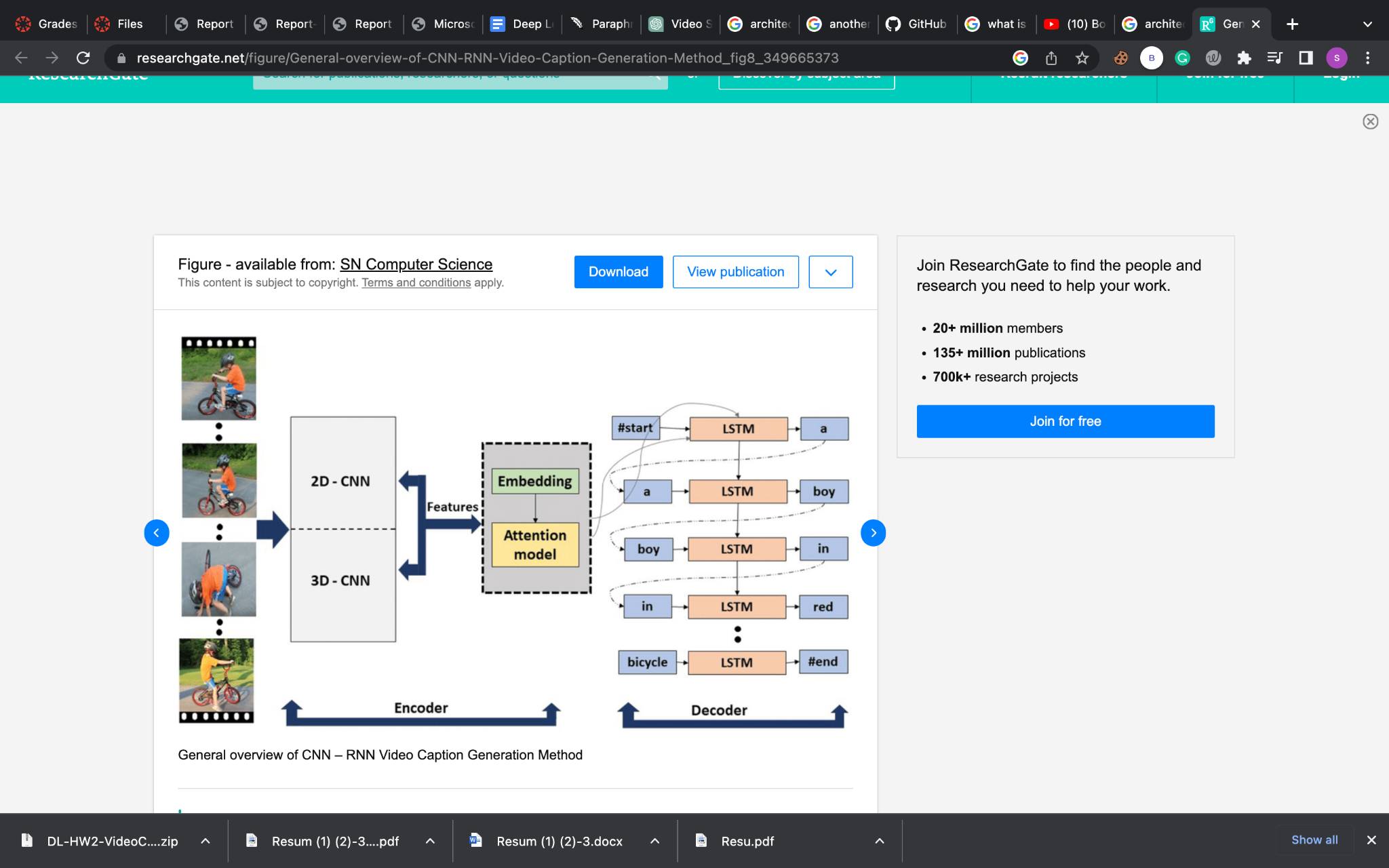
**Requirements:**

For this Caption Generation, The Requirement tools that I have used are primarily python's latest version-3.9.7, and I also used the Cuda application programming interface and NumPy, Pandas Dictionaries. Another TensorFlow is a software library for machine learning and artificial intelligence. The above are the requirements I have used for this creation of Captions.

**Approach:**

In our model, we have considered the sequence-sequence approach in that we will give the video frames to priorly trained CNN VGG19. It creates an 80×4096-sized spatial feature map of all the Videos. This model takes a sequence of frames from the videos in the MSVD dataset and produces a sequence of Captions as Output.

**Long Short-Term Memory(LSTM) Networks for Encoding and Decoding:**

****

The above figure shows the Creation of Captions From Video frames By using the RNNs and CNN and LSTM. The initial encoding stage after the next Stage is decoding. To decode a natural language description of the characteristics seen, the full input sequence is first encoded, condensing the video into a single hidden state vector.

**Attention Model:**

The decoder obtains input from each hidden state of the encoder by utilizing the attention mechanism, which acts as an interface connecting the encoder and decoder. During each step of the decoding process, the model can examine multiple segments of input data by attention mechanism.

**Bilingual Evaluation Understudy(BLEU) Score:**

The BLEU is an algorithm and a statistic for automatically assessing text that has been machine translated, and also used for assessing the accuracy of text that has been computer translated between two natural languages. The principle behind BLEU is that a machine translation is considered good if it closely resembles a professional human translation. Quality is evaluated by comparing the machine's output to a person's. BLEU was one of the first measures to correlate with human assessments of translation quality strongly and is still a popular and cost-effective automated metric used today.

Source: Wikipedia

**Schedule Sampling:**

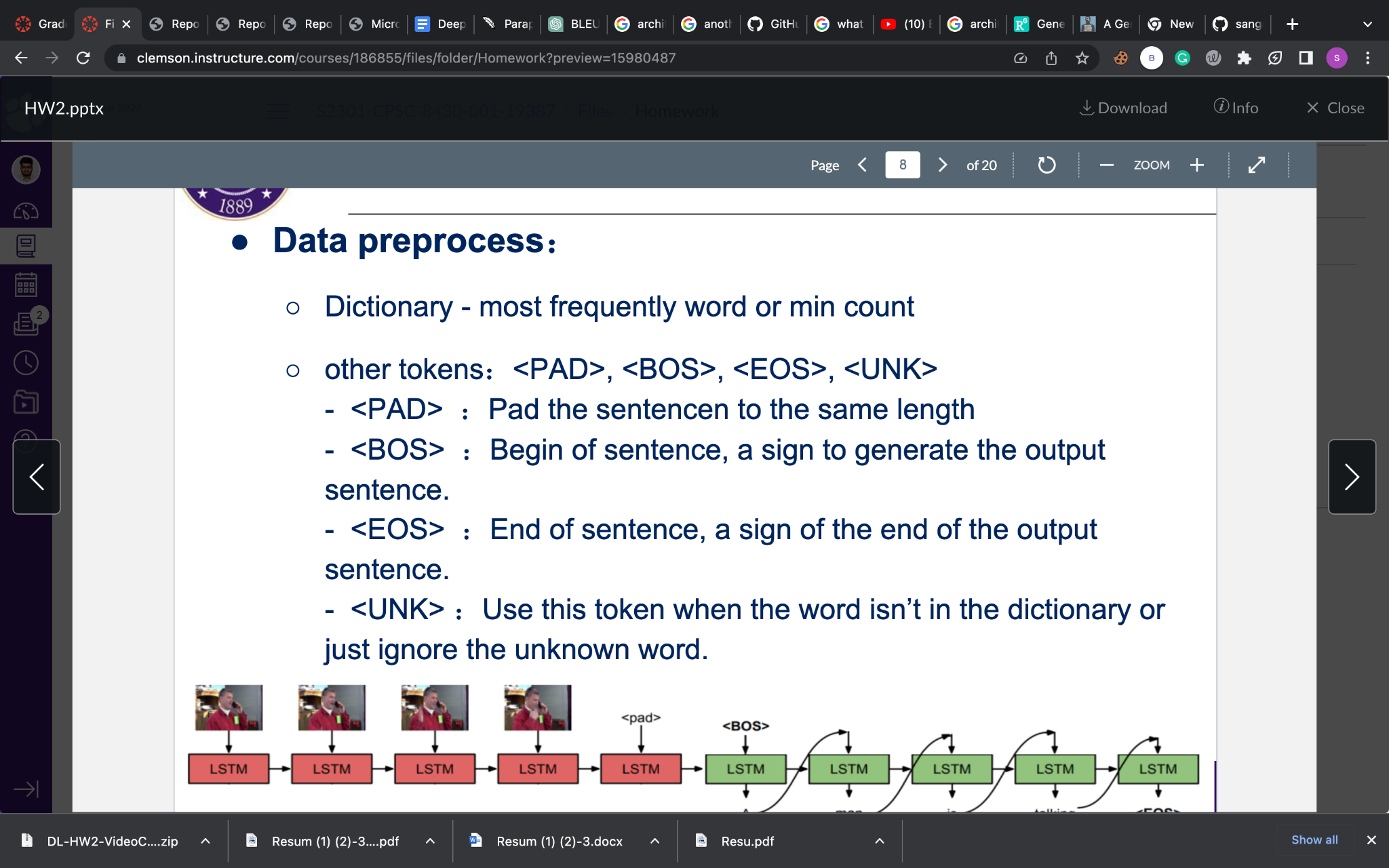
The model substitutes a token it creates for an unknown previous token during the inference phase. With a method called scheduled sampling, which randomly replaces specific discrete units from the past with the model's prediction, the difference between training and testing timeframes is reduced. This aids in addressing the model's exposure bias problem.

**Execution Steps:**

During pre-processing, we add padding and markers to the data to process only non-padded parts of the input sentence using RNN through packed padded sequences. We also use masking to ignore irrelevant elements such as attention over padded elements to improve performance. The training and testing data were obtained from the PowerPoint presentation and stored locally in the 'MLDS\_hw2\_1\_data' folder. The minimum vocabulary size required is three.

**Tokenization:**

The following are four tokens used in natural language processing:



To create a dictionary containing the IDs and captions of each video, two object files are generated. These files are named 'vid\_id.obj' and 'dict\_caption.obj'. Additionally, another object file called 'dict\_feat.obj' is created as part of the dictionary-building process.

**To run sequence.py on the Palmetto cluster using my active node, I utilized the following command.**

/Users/sandeepvarmasangaraju/Downloads/MLDS\_hw2\_1\_data/testing\_data/feat /Users/sandeepvarmasangaraju/Downloads/MLDS\_hw2\_1\_data/testing\_label.json

Below are the results I got for Sequence.py execution.

**#/Users/sandeepvarmasangaraju/Downloads/MLDS\_hw2\_1\_data/training\_data/feat**

**#./Users/sandeepvarmasangaraju/Downloads/MLDS\_hw2\_1\_data/training\_label.json #./output\_testset\_sandeepvarmasangaraju.txt**

Among 6098 words filtered, 2881 words from the dictionary with a minimum count of 3

Caption dimension: (24238,2)

Captions max length: 40.

The average length of captions: is 7.711094526242027

Unique tokens: 6447

The ID of the video: is EPXsiQw9vvo\_1\_12.avi

The shape of features of video: (80,4097)

The caption of the video: the target video got more bullet holes.

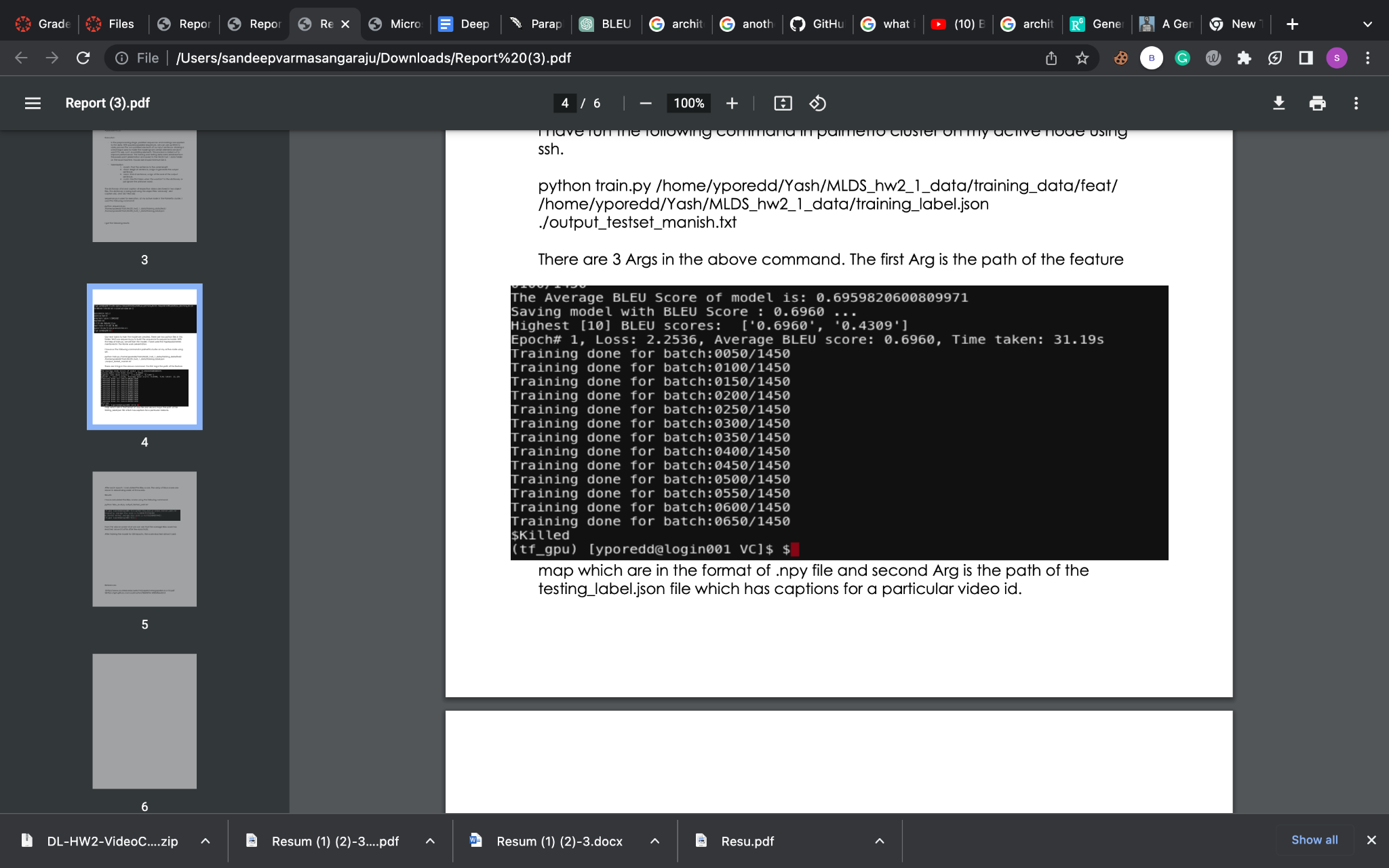
(tf\_gpu) [sandeepvarmasangaraju VC] $

* Our aim now is to train the sequence-to-sequence model we constructed using two Python scripts: sequence.py, which builds the model, and train.py, which trains it. We used hyperparameters to fine-tune the learning process, and we have included a table listing the hyperparameters utilized in our experiment.Model hyperparameters, such as the neural network's topology and size, and algorithm hyperparameters, such as the learning rate and batch size, are examples of hyperparameters.
* Hyperparameters are parameters that help regulate the learning process during model training. They can be categorized into model hyperparameters, which determine the neural network's size and topology, and algorithm hyperparameters, which include learning rate and batch size. These hyperparameters regulate the learning process This may have a substantial impact on the effectiveness and performance of the model. Some of the Hyperparameters are the learning rate, Use\_attention, Max\_encoder\_steps, and Max\_decoder\_steps.

I utilized Secure Shell (SSH) to execute a command on the active node of the Palmetto cluster.

**#/Users/sandeepvarmasangaraju/Downloads/MLDS\_hw2\_1\_data/testing\_data/feat**

**#./Users/sandeepvarmasangaraju/Downloads/MLDS\_hw2\_1\_data/testing\_label.json #./output\_testset\_sandeepvarmasangaraju.txt**



Here are the specifications and evaluation metrics of the model:

1. The model has undergone 225 epochs during the training process.

2. It was trained with a learning rate of 0.0001, using batches of 128 items.

3. The model has 512 hidden layers.

4. The Adam optimizer was employed, with a dropout rate of 0.3.

5. This model has a teacher-learning ratio of 0.7.

6. The vocabulary size was set to n greater than 4.

I computed the Bleu score after every iteration and then sorted the scores in descending order.

**Results:**

To obtain the results, I used the command Below

"Python bleu\_eval.py output\_testset\_sandeepvarmasangaraju.txt"

to calculate the Bleu scores. The original average score was 0.2689437917016406, while the average score obtained by another method was 0.6959820600809971.

The results show that the average Bleu score increased to around 0.7121 after multiple

epochs of training the model. The score almost reached 0.7034 after 250 epochs.

Python dictionaries were used to implement the different data types. The frequency of each word in the training label file was recorded and used to create the vocabulary using the "word dict" dictionary. The "w2i" dictionary stored every word with a frequency below 4, and each word was mapped to its corresponding index. The "i2w" dictionary stored the reverse mapping of indexes to the corresponding words in the vocabulary.