THE PENNSYLVANIA STATE UNIVERSITY
SCHREYER HONORS COLLEGE

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Leveraging Transformer Models to Enhance Temporal Grounding

JASON WANG
SPRING 2022

A thesis
submitted in partial fulfillment
of the requirements
for a baccalaureate degree
in Computer Science
with honors in Computer Science

Reviewed and approved* by the following:

Rui Zhang
Assistant Professor in Computer Science and Engineering
Thesis Supervisor

John Hannan
Associate Department Head and Associate Professor of Computer Science and Engineering
Honors Adviser

* Electronic approvals are on file.

ABSTRACT

Due to the exponential growth of video in all our lives, the goal to understand details and actions within videos has never been more important. When words and sentences are grounded into images and videos, they become far more meaningful. Incorporating both Computer Vision (CV) and Natural Language Processing (NLP), the task of temporal grounding aims to predict a specific range of time that an event happens within a video. Specifically, temporal grounding takes a natural language query and an untrimmed video as input. Tackling and ultimately optimizing this task can open a wide range of applications both in NLP and CV. From detecting actions and objects in a live video to creating unsupervised captions, temporal grounding has an abundance of benefits. In this thesis, I first recap the innovations presented in EVOQUER, a temporal grounding framework we created that incorporates an existing text-to-video grounding model and a video-assisted query generation network. Afterwards, I present and analyze the potential benefits of leveraging transformer models as well as our current attempts at replicating previous performance statistics.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I want to thank Professor Rui Zhang and Dr. Yanjun Gao for their continued mentorship and guidance throughout my research journey. I would also like to thank my teammates Michael Chan and Lulu Liu for their continued work on this project. Lastly, I want to thank my friends and family for their constant support in my education and endeavors.

Chapter 1

An Introduction

1.1 Background

Temporal grounding aims to find the time interval in an untrimmed video that expresses the same meaning as a natural language query. It addresses the temporal, semantic alignment between language and vision by locating the video content that corresponds to a query. These two inputs are visualized in Figure 1 below. From the corresponding query "Spread mayonnaise on bread," the temporal grounding model must understand the meaning of both the query and input video while ignoring irrelevant actions. The correct time interval is oftentimes known as the ground truth and is labeled in orange. This process of learning the meaning of a query falls under the scope of NLP, while the process of understanding the video falls under CV. Thus, temporal grounding is a task that incorporates two ever-growing fields. It is broadly applicable in many tasks such as visual storytelling [1,2], video caption generation [3], and video machine translation [4]. In the real-world, optimizing temporal grounding can increase the safety of homes and cars through action and object detection. It can also help visually impaired individuals understand videos through generating captions without human assistance. With so many more potential applications, temporal grounding is a critical step into merging language and vision.

In recent years, work on temporal grounding has achieved significant progress through a wide range of approaches. Some methods emphasize mapping verbs and nouns to visual clues such as actions and objects [5]. Others utilize RGB data from images as well as optical flow for actions [23]. For those unfamiliar, optical flow is the movement of objects within a scene from a certain point of view. This movement helps models understand the speed at which objects are moving. Despite this promising progress, many methods are limited by only employing a one-directional, single-task learning flow. This

learning flow means that the predicted time intervals are not used to optimize learning. Furthermore, this method can not only slow training, but also limits understanding. To strengthen learning, I looked to explore the possibility of enhancing temporal grounding through these related works. With the recent growth of transformer models and the state-of-the-art performance many frameworks with transformers exhibit, leveraging transformer models could be the next step into enhancing temporal grounding.

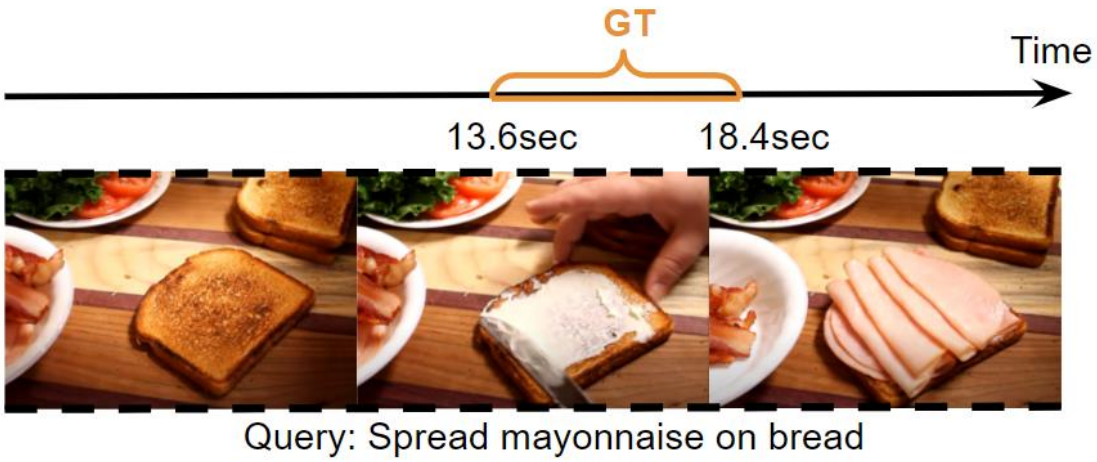

Figure 1. Time Interval in Video Example

1.2 Related Work

Previous works on temporal grounding can be split into three categories: reinforcement learning, weakly supervised, and strongly supervised. The most prominent example of a strongly supervised approach is the LGI algorithm [5]. Utilized within the EVOQUER framework described later in this thesis, LGI achieves state-of-the-art performance on both the Charades-STA and Activity Net datasets. These datasets will be described in detail later in this work as well. In this context, the term strongly supervised means that the input videos have previously been described by a ground truth. In contrast, weakly supervised means that input videos are not trained using a ground truth. Thus, datasets for strongly supervised approaches are typically very difficult to create, requiring volunteers to manually record their actions. In the case of the LGI algorithm and many other strongly supervised approaches, word-level and sentence-level attention is used to predict time intervals within a video. For clarity, attention means the level of emphasis a model should put onto certain words and sentences. To this end, the state-of-the-art LGI algorithm was utilized within the EVOQUER framework to create this sentence-level attention. Instead of the model just knowing the relationship between words in a single query, sentence-level attention allows the model to learn the ordering of multiple sentences. This is especially important for long, complex sentences where a handful of examples can contain a wide range of unique words that may not appear in other sentences within the dataset.

Another task similar to temporal grounding is text-to-video moment retrieval, which focuses on the grounding between a query and video. Instead of the strongly supervised approaches described above, weakly supervised approaches have been developed for text-to-video moment retrieval [20]. Both tasks require grounding language into vision while analyzing the understanding of natural language. Similarly, temporal grounding also has its roots in other tasks such as video captioning, which aims to generate a description of text given a video. Having many related tasks, temporal grounding also shares the issues related with captioning and retrieval. Since videos can sometimes consist of hundreds if not thousands of frames, determining important frames can be a significant road-block. Despite the copious amount of

video data, many previous models suffered slow training times and inconsistent results. With the creation of transformer models such as BERT [11], models that use end-to-end transformers for video-captioning have been developed. Furthermore, due to these promising results, it is logical to think transformer models could be leveraged to achieve better performance on related tasks, particularly temporal grounding.

1.3 Metrics & Definitions

In this and previously referenced works, two main conventional metrics for temporal grounding are adopted: 1) R@tIoU which determines the recall at different thresholds (Specifically 0.3, 0.5, and 0.7); and 2) mIoU which reports the average recall from all three threshold levels. For the former, R@tIoU stands for the mean intersection over union. Commonly used for semantic image segmentation [21], R@tIoU determines whether there is a certain percent overlap (30%, 50%, and 70%) between the ground truth and predicted output. For the case of semantic image segmentation, images are split into distinct classes. Since all frameworks described later in this work are strongly supervised, this metric is the primary way we benchmark performance. This ratio is visualized for clarity in Figure 2. The red area highlights the value for which the metric R@tIoU measures.

One of the key novel ideas of the EVOQUER framework is query simplification. Due to the usage of this intuition, there are additional metrics we utilize to determine performance as well. Specifically, predicted queries are evaluated with two metrics: 1) Jaccard similarity; and 2) BLEU score. Like R@tIoU, Jaccard similarity measures intersection over union between words in ground truth and in prediction. Since it does not penalize for duplicated words, Jaccard similarity gives a rough estimation for the quality of the translated output. Despite being very similar to R@tIoU, Jaccard similarity is pivotal due to the increase in likelihood that words may be duplicated. Since query simplification decreases the number of unique words by removing irrelevant adjectives and adverbs, nouns and verbs make up most of

the dictionary. Because there are less words to predict, it is intrinsically more likely that a model will perform better since it is more likely that correct words will be predicted. Thus, Jaccard similarity takes this into account, alleviating potential incorrect and misleading performance improvements. Lastly, BLEU score is a standard evaluation metric for machine translation that measures n-gram word overlap. Since the simplified queries are two words in length, BLEU score is reported in both unigram and bigram.

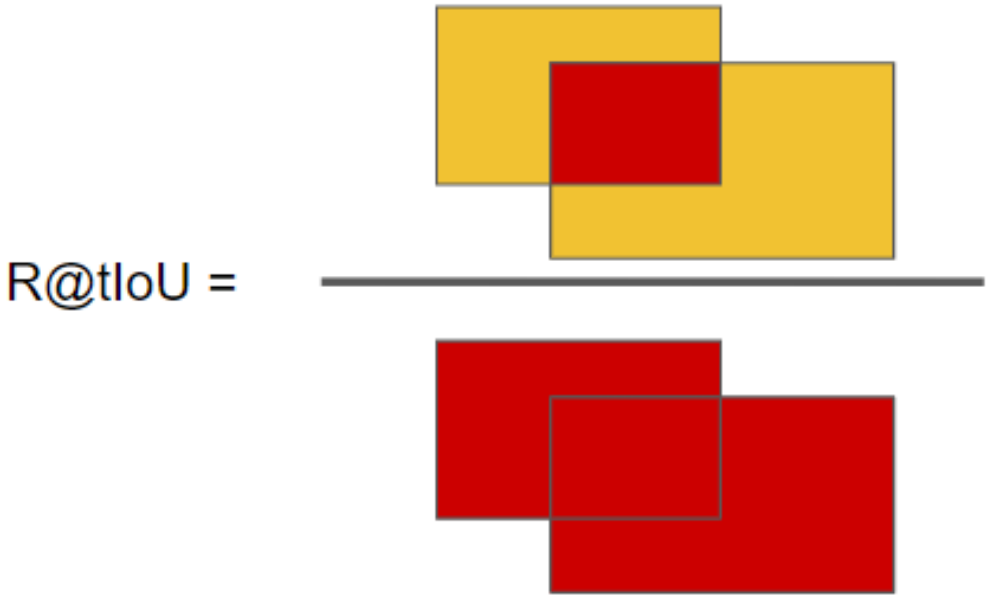

Figure 2. Intersection over Union Visualization

Chapter 2

The EVOQUER Framework

2.1 Closed-Loop Framework

The EVOQUER Framework, standing for Enhancing Temporal Grounding with VideOPivoted Back QUERy Generation, was a model developed by Yanjun Gao, Lulu Liu, myself, and Professor Zhang to tackle temporal grounding. One of the key intuitions that EVOQUER incorporates is the idea of feedback-error-learning from control theory [7,8]. In short, closed-loop learning allows for better training by allowing the model to learn from its mistakes for efficiency. The control network in the EVOQUER framework learns to correct its error from feedback and gains stronger supervision to increase learning. In this context, the control network simply means the fixed loop that the inputs and outputs of the model are pushed through. To obtain this feedback, EVOQUER uses a closed-loop framework for temporal grounding that receives two important enhancements: 1) Supervision in predicting time intervals; and 2) Feedback from the output video features extracted from the prediction. These inputs are represented as arrows in Figure 3.

To achieve this, EVOQUER involves two components: 1) An existing temporal grounding module; and 2) A translation module. First, the temporal grounding module predicts time intervals given an untrimmed video and a query as described above. Afterwards, the translation module takes queries by the predicted intervals as input and outputs a simplified query with only verbs and nouns. As shown by Figure 3, both tasks receive the same inputs but output different results. To complete query simplification as described above, EVOQUER uses the video machine translation framework VMT [9]. Standing for Video-guided Machine Translation, VMT was originally used for language translation from English to Chinese. In contrast to previous frameworks developed for translation tasks, VMT utilizes information from videos. Specifically, I3D features from a 3D ConvNet are used to assist the framework in predicting

more accurately. Since we already create I3D features for video representation in the temporal grounding module, this was an intuitive addition to the closed-loop framework. In addition to ease of use, EVOQUER also utilizes this framework due to its proven performance benefits in video-assisted bilingual translation. Although we do not focus on language translation, VMT shares a similar Encoder-Decoder framework that other temporal grounding models utilize as well.

Since the creation of EVOQUER, there have been many other improvements to video-guided machine translation as well [22]. In the case of [22], positional encodings are utilized to achieve the optimal BLEU score. A formal description of what position encodings are is included in Chapter 3 below.

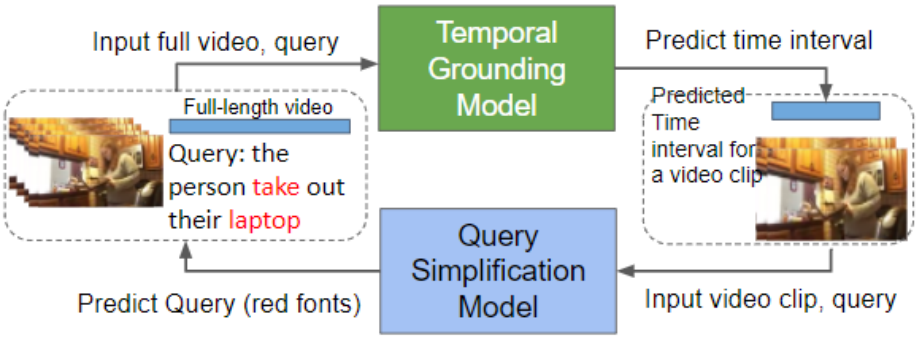

Figure 3. Closed-Loop Framework

2.2 Pipeline Overview

An overview of the EVOQUER pipeline can be found in Figure 4 on the following page. As shown by the box on the top left, the input to the framework is an untrimmed video and a set of English queries. Following the framework in LGI [5], EVOQUER uses I3D frame-based features for video representation. Since LGI shows state-of-the-art performance using these video features, we continue with this approach. Given the video features and queries, LGI predicts intervals with the content corresponding to a given query. With this output, we exit the temporal grounding module labeled in green.

Following the creation of these time intervals, frames are extracted from videos trimmed by the predicted interval to represent the contents of the video clip. To maintain the continuity of the contents, EVUOQUER extracts 32 frames per video clip such that that the contents of the trimmed videos are evenly distributed across all 32 frames. This spacing lessens the likelihood that a set of actions happens during a very short interval within the input video. Within the datasets we use, videos are captured at 24 frames per second. Thus, we found the 32-frame videos roughly to span 1.3 seconds.

Next, we enter the translation module within the EVOQUER framework. First, we feed the extracted video features and input query into the translation module consisting of two bi-LSTM-based encoders and an LSTM-based decoder. Following the output of the encoders, temporal and soft attention are used for the input to the decoder. As the name implies, temporal attention puts emphasis on certain time intervals with a video. Additionally, soft attention utilizes a differentiable function to determine which words within a sentence should be emphasized. An important distinction is the use of hard vs soft attention. Rather than the continuous weighted averages used in soft attention, hard attention utilizes discrete values to determine whether a specific word or frame should be attended to. Since hard attention can completely mask certain words, it is best to utilize soft attention in this context. Video hidden states and text hidden states are sent individually to these two attention modules. This output is then concatenated into a one-vector form and sent to the decoder as input. In the attention network, temporal attention is learned through video features, and soft attention through query hidden states.

Instead of learning to decode the original query, EVOQUER tries to focus on verbs and nouns, since that is what distinguishes the video content. In the Charades-STA dataset, annotators tend to use various verb tenses when describing the video activities. For example, both “closing the window" and “closes the window" are used on the same video content. Therefore, we lemmatize the words, label the query with part-of-speech tags, and extract verbs and nouns as simplified versions of the queries. The decoder then learns to predict simplified queries and computes a negative log-likelihood (NLL) loss at the end of the decoding. Finally, EVOQUER combines the NLL loss from query simplification with the LGI loss [5] from time interval prediction to train the networks jointly in an end-to-end fashion.

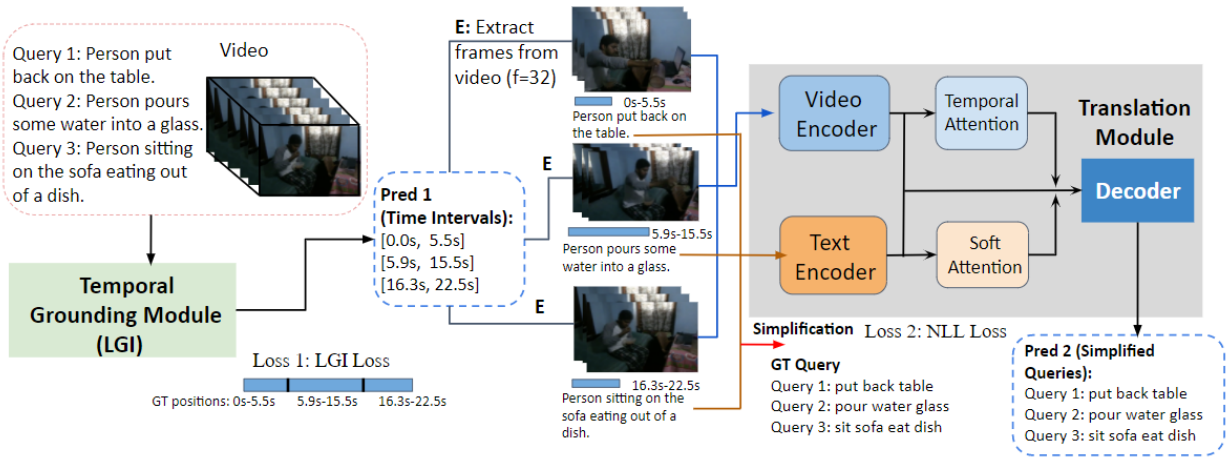

Figure 4. EVOQUER Pipeline Overview

2.3 Benchmark Datasets

The two datasets used to evaluate the EVOQUER framework are Charades-STA and Activity Net, two widely used benchmark datasets for temporal grounding. We follow the dataset settings in [5], where both datasets are set with train/valid/test as 50%, 25%, and 25% respectively. The statistics of each of these datasets are detailed in Table 1 below. As shown evidently from the table, the two datasets vary greatly on most of the statistics. Compared to Charades-STA, Activity Net is a more challenging dataset since it requires the decoder in EVOQUER to predict correct words from a much larger vocabulary. A dictionary whose size is almost 10 times larger than Charades-STA.

Due to these stark differences, I created two distinct data loaders to prepare the data for training. The code of the data loader for Activity Net can be found in the Appendix below. In the case of Charades-STA, the given input format included a video id, time stamp, and query. An example of this input is shown in Figure 5. To input this data into the neural network, I simplify the data into a list of video ids, timestamps, duration, and query length. To compare the outputs of the model to the ground truth, I create PyTorch tensors with data regarding the ground truth, including start position, end position, and the number of video features. Each video id corresponds to a specific video, enabling the neural network to differentiate each input while training. Lastly, I create two dictionaries to create a mapping from words to ids and vice versa. In contrast to Charades-STA, Activity Net had a far larger dictionary, making the dataset far harder to simplify. Specifically, I had to simplify how many inputs were provided in each instance. In this context, an instance was a dictionary of all the inputs associated with one video id. One method I utilized was to ignore inputs such as description length and description labels since these inputs were unique to Charades-STA. Following the development of these two data loaders, EVOQUER was tested on both.

In the experiments on Activity Net, EVOQUER was found to converge at a higher NLL loss than Charades-STA, and failed to produce good quality simplified queries. We believe this was likely due to

the harder decoding task. I later detail another dataset when transformers are introduced. Specifically, we utilize the HowTo100M dataset and focus solely on a subset of cooking videos.

Dataset	Charades-STA	ActivityNet
Num Queries	27,847	71,957
Num Videos	9,848	20,000
Avg Video Len (Sec)	~30	~120
Input Query |V|	1,140	11,125
Simpl. Query |V|	560	5,946
Simpl. #Tks per Query	2.31	4.12

Table 1. Statistics on Benchmark Datasets

```
N14BK 39.3 45.0##person put a notebook in a bag.
NQT1S 18.3 31.0##person all of a sudden they start sneezing.
F1VEE 27.0 32.0##person opening a door.
F1VEE 15.1 23.8##person they put on their shoes.
F1VEE 15.1 23.8##person puts on shoes.
F1VEE 19.2 28.2##person they take a mobile phone.
F1VEE 15.1 23.8##person put some shoes on.
YDWN5 6.2 12.6##a person tidying a wardrobe in an entryway is smiling.
QRWQ3 19.1 29.0##a person cooking on the stove.
QRWQ3 19.1 29.0##a person is cooking something on a stove.
SFHHR 21.5 28.0##the person puts the coffee on the table.
HWYTN 0.0 4.0##another person runs out the room.
HWYTN 0.0 4.0##another man running past.
HWYTN 0.0 4.0##another person is running in shoes.
```

Figure 5. Data Loader Input Example

2.4 Results & Analysis

Table 2 presents results on the Charades-STA and Activity Net from two models: 1) The EVOQUER model; and 2) A retrained LGI model. Compared to the LGI framework, EVOQUER showed improvement on R@0.7 and mIoU, the hardest thresholds for temporal interval overlap. Table 3 divides the samples into four categories according to their recall: 1) When EVOQUER ranked in a higher threshold than the LGI; 2) When EVOQUER ranked lower than the LGI; 3) When both had the same recalls that were at least R@0.3; and 4) When both scores ranked below R@0.3. In contrast to the 441 samples on Charades-STA, Activity Net had 4,268 samples above R@0.3, with 79 and 1,144 samples of absolute improvement. These statistics can be found in further detail in Table 3. Despite a stark increase in samples between the two datasets, the ratio between samples above and below R@0.3 remained the same. Thus, this continued performance showed that EVOQUER had promising results on a wide range of potential benchmark datasets. In addition to mean intersection over union, EVOQUER achieved 51.98 on Jaccard Similarity, 53.04 on BLEU unigram, and 42.47 on BLEU bigram. Compared to the state-of-the-art model of LGI, EVOQUER performed very similarly. In general, EVOQUER showed incremental improvement in these metrics, despite incorporating query simplification on Charades-STA.

Through a case study, EVOQUER demonstrated the power of query simplification when used. Utilizing query simplification on Activity Net, we experienced some challenges. Specifically, we found performance improvements from query simplification to be less noticeable on Activity Net than Charades-STA. We accredited these shortcomings to two main reasons: 1) he decoder vocabulary on Activity Net was much larger than Charades-STA, making sentence simplification less consistent; and 2) The design of translation module was too simple to handle longer predicted time intervals. As shown from Table 1, the videos of Activity Net proved to be far longer than Charades-STA. Rather than the 30 seconds videos of Charades-STA, Activity Net could include videos of upwards of multiple minutes. This increase in video length, coupled with the large vocabulary, likely made the model struggle with

predicting the ground truth. Nonetheless, the results on Charades-STA indicated the contributions of the EVOQUER framework.

Data	Model	R@0.3	R@0.5	R@0.7	mIoU
char	LGI	71.54	**58.08**	34.68	50.28
	EVOQUER	**71.57**	57.81	**35.73**	**50.48**
anet	LGI	57.76	40.38	22.60	40.65
	EVOQUER	**59.21**	**42.02**	**23.91**	**41.61**

Table 2. Results of EVOQUER Framework vs LGI

	Both >= R@0.3			Both
	EVOQUER ↑	EVOQUER ↓	Same	<R@0.3
char	441	362	1,347	777
anet	4,268	3,124	8,074	10,538

Table 3. Charades-STA vs Activity Net Sample Distribution

Chapter 3

Leveraging Transformers for Temporal Grounding

3.1 Why Transformer Models

First introduced in 2017 [10], transformers rose in popularity due to being easy to train and efficient in tasks previously dominated by Long Short-term Memory Networks (LSTMs). Since LSTMs had to read inputs sequentially, they were not only difficult to train on large datasets but were also ineffective for long streams of sequential data. In the case of transformers, substantial amounts of sequential data, such as frames that make up a video, can be trained in parallel. Due to this ability to parallelize data, transformers were also able to align with modern GPU architectures as well.

The two biggest innovations provided by transformers are self-attention and positional encodings, two aspects that can benefit temporal grounding. Originally used for sentence translation, such as translation from English to French, transformers must find different ways to encode the position of different words within a sentence since transformers do not take inputs sequentially like LSTMs. Furthermore, the order of words is lost when inputting the data into the model. The importance of the order of words can be shown easily through the following example: 1) I genuinely need to finish this interview; and 2) I need to finish this interview genuinely. Despite both sentences containing the same words, the placement of words such as genuinely change the meaning of the sentence completely. Prior to inputting embeddings into an encoder and decoder framework, transformers encode a sin and cos function into the embeddings depending on a word's position in a sentence. This process of positional encodings was also utilized in other tasks such as the video-guided machine translation described earlier as well.

In addition to positional embeddings, transformers utilize self-attention. In short, self-attention is used to obtain context by finding associations between different words in the same input. In contrast to simple attention which uses emphasis learned from external inputs. Moving forward, I talk about previous

applications of transformer models: first in the context of NLP and then in the context of temporal grounding.

3.2 Previous Work

As a variation of the Encoder-Decoder framework used in the original paper [10], BERT [11], also known as a Bidirectional Encoder Representation from Transformers, was created. BERT differentiates itself by stacking the encoder modules in [10] to broaden the application of the initial framework. To pretrain the model, BERT utilizes two main problems to allow the model to understand language: 1) Masked LM (MLM); and 2) Next Sentence Prediction (NSP). For the former, roughly 15 percent of the inputs are randomly replaced by masked tokens, and the model is required to predict the missing word. For the latter, two sentences are provided as input, and BERT makes a prediction on whether one sentence follows another. These tasks for pretraining help BERT associate not only sentences, but also the relationship between words in two directions. By changing the input and output layers before and after the model, BERT can be modified for a very wide range of NLP tasks [12,13,14].

Following the work of BERT, VideoBERT[12] was developed to extend the power of transformers on vision-related tasks. Instead of masked sentences, VideoBERT is inputted a concatenated sentence with visual words. These novel visual words are created with features which capture the visual frames most representative of the video. Similar to the NSP task utilized to pretrain BERT, VideoBERT utilizes a linguistic-visual alignment task that predicts if the sentence corresponds with the visual words described above. Due to the versatility of BERT on a wide range of NLP tasks, VideoBERT has similarly been used for a wide range of vision and language tasks [15, 16]. By changing the output layers of VideoBERT, leveraging transformer models to predict time intervals instead of videos can enhance temporal grounding. With these potential benefits, we investigate adding additional layers and/or utilizing previous works to increase performance.

3.3 Attempts at Replication

Since we do not have access to a pretrained VideoBERT model, replicating the VideoBERT framework has been our ongoing task. Of the available options, we use a subset of the HowTo100M dataset [18] to align with the original YouCook II dataset. Specifically, we focus on roughly 16,000 cooking-related videos to narrow the focus of our model to a subset of tasks. Downloaded from Youtube, the HowTo100M dataset emphasizes instructional videos as well, allowing each video to have a specific number of intended actions. Thus, rather than videos with sporadic actions scattered across the video, datasets such as HowTo100M and YouCook II contain videos best for training the neural network.

To replicate the performance of VideoBERT, we first created I3D features [19] of each video. Standing for Inflated 3D ConvNet, I3D features obtain the spatio-temporal understanding of each video. For much of this replication, we treat I3D as a black box. Compared to the S3D features [17] utilized within VideoBERT, the I3D framework creates features with dimensions 20 by 600. In this context, we interpreted this dimensionality as each of the 20 frames having 600 specific features. Within [12], VideoBERT utilizes S3D features with 1024 features per frame. Since this difference in dimensionality has not caused any issues during training, we moved on to create corresponding centroids for each video feature. On average, we found that each video we focused on in the HowTo100M dataset was caught at 25 frames per second. Thus, each video had an average of 260 frames. This large number of frames relates more closely with Activity Net rather than Charades-STA.

Using these feature files, we utilized mini-batch k-means to create centroids for each video. Specifically, we utilize hierarchical k-means clustering from sklearn, a built-in machine learning library for Python. By default, each centroid had dimensionality 12 by 600 where 12 represented the number of frames, and 600 represented the number of features per frame. One of the key roadblocks we faced was this difference in dimensionality between the created I3D features and centroids. To solve this issue, we created two potential solutions: 1) Adjust the parameters to the hkmeans script from sklearn to try to modify the outputted dimensionality; and 2) Adjust the centroids to 1 by 600 by taking the average of

each of the 12 frames. This solution removes the possibility of dimension-errors moving forward. Using the second solution, we take the average of each centroid and continue with the pipeline.

Utilizing these centroids, we concatenate them for ease of use. Next, we utilize this large centroid to label and punctuate the data prior to training. Currently, we are in the process of training the model with some initial success. Figures 6 and 7 below show a consistent decrease in joint loss as training continues. Since VideoBERT is trained on text-only, video-only, and joint text and video, we graph joint loss to show an aggregate metric of progress. Despite this decreasing trend, we find that the loss starts to stagnate at roughly iteration 600. Since this is still early in the training process, it is difficult to pinpoint the exact cause of this stagnation. We hope to train our model on at least 10 epochs to achieve better performance.

Following a fully trained VideoBERT model on the HowTo100M dataset, we plan to edit the output layers of VideoBERT to leverage its performance on temporal grounding. Similar to EVOQUER, testing the performance of this novel framework on benchmark datasets such as Charades-STA and Activity Net is the true tell of whether a transformer architecture can benefit this task. Due to previous success in extending VideoBERT to tasks such as predicting video sequences, leveraging transformers could be the next significant move towards enhancing temporal grounding.

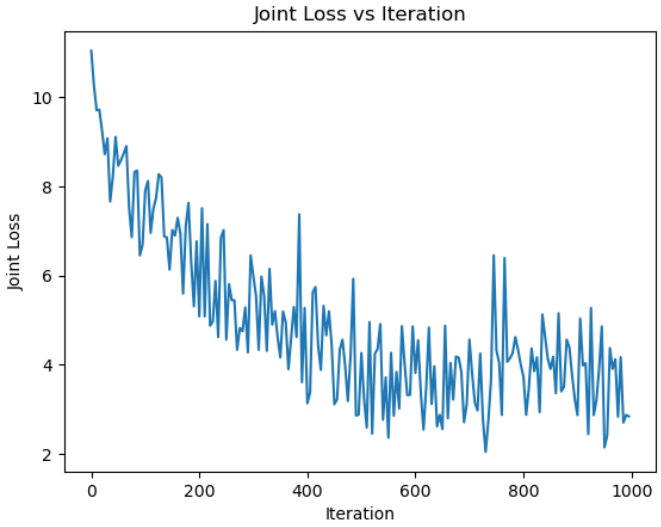

Figure 6. Joint (Both Text & Video) Loss vs Iterations During Training

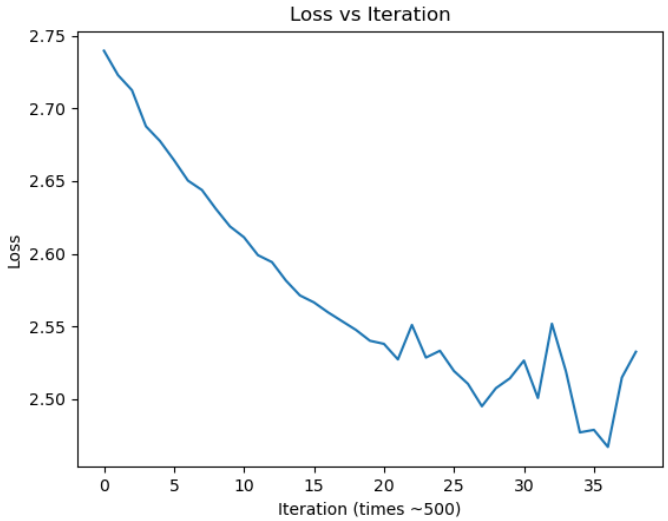

Figure 7. Joint Loss vs Every 500 Iterations

Appendixes

Appendix A Data Loader Code

```
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

import os
import h5py
import string
import numpy as np
import string
import torch
import json
import time

import torch
import torch.utils.data as data

from collections import defaultdict, OrderedDict
from tqdm import tqdm
from abc import abstractmethod
from src.dataset.abstract_dataset import AbstractDataset
from src.utils import utils,io_utils

def create_loaders(split, loader_configs, num_workers):
        dsets, L = {}, {}
        for di,dt in enumerate(split):
                shuffle = True if dt == "train" else False
                drop_last = True if dt == "train" else False
                dsets[dt] = ActivityNetCaptionsDataset(loader_configs[di])
                L[dt] = data.DataLoader(
                        dsets[dt],
                        batch_size = loader_configs[di]["batch_size"],
                        num_workers = num_workers,
                        shuffle = shuffle, # shuffle
                        collate_fn = dsets[dt].collate_fn,
                        drop_last= drop_last #drop_last
                )
        return dsets, L

def __getitem__(self, idx):
                # get query id and corresponding video id
                qid = str(self.qids[idx])
                vid = self.anns[qid]["video_id"]
                timestamp = self.anns[qid]["timestamps"]
                duration = self.anns[qid]["duration"]
                qry=self.anns[qid]["query"]
                # Descriptions
                #d_label_lsts = self.descriploader.encoded_description(vid, self.wtoi)
                #description_length, description_labels =
d_label_lsts['descrip_lengths'], d_label_lsts['descrip_labels']

                # get query labels
                if self.in_memory:
```

```
                        q_label = self.query_labels[qid]
                else:
                        query_labels = h5py.File(self.paths["query_labels"], "r")
                        q_label = query_labels[qid][:]
                q_leng = self.query_lengths[qid]

                # get grounding label
                if self.in_memory:
                        start_pos = self.s_pos[qid]
                        end_pos = self.e_pos[qid]
                else:
                        grd_info = h5py.File(self.paths["grounding_info"],"r")
                        start_pos = grd_info["start_pos/"+qid][()]
                        end_pos = grd_info["end_pos/"+qid][()]

                # get video features
                if self.in_memory:
                        vid_feat_all = self.feats[vid]
                else:
                        vid_feat_all = io_utils.load_hdf5(self.feat_hdf5,
                    verbose=False)[vid]["c3d_features"]
                vid_feat, nfeats, start_index, end_index = self.get_fixed_length_feat(
                                vid_feat_all, self.S, start_pos, end_pos)

                # get video masks
                vid_mask = np.zeros((self.S, 1))
                vid_mask[:nfeats] = 1

                # get attention mask
                if self.in_memory:
                        att_mask = self.att_mask[qid]
                else:
                        att_mask = grd_info["att_mask/"+qid][:]
                instance = {
                        "vids": vid,
                        "qids": qid,
                        "timestamps": timestamp, # GT location [s, e] (seconds)
                        "duration": duration, # video span (seconds)
                        #"description_length": description_length,
                        #"description_labels": description_labels,
                        "query_lengths": q_leng,
                        # "query":qry,
                        "query_labels": torch.LongTensor(q_label).unsqueeze(0),
                        "query_masks": (torch.FloatTensor(q_label)>0).unsqueeze(0), #
[1,L_q_max]
                        "grounding_start_pos": torch.FloatTensor([start_pos]), # [1];
normalized
                        "grounding_end_pos": torch.FloatTensor([end_pos]),     # [1];
normalized
                        "grounding_att_masks": torch.FloatTensor(att_mask),  # [L_v]
                        "nfeats": torch.FloatTensor([nfeats]),
                        "video_feats": torch.FloatTensor(vid_feat), # [L_v,D_v]
                        "video_masks": torch.ByteTensor(vid_mask), # [L_v,D-v]
                }

                return instance

def collate_fn(self,data):
            seq_items=["video_feats","video_masks","grounding_att_masks"]

tensor_items=["query_labels","query_masks","nfeats","grounding_start_pos","grounding_e
nd_pos",]
```

```
            batch={k:[d[k] for d in data] for k in data[0].keys()}
            if len(data)==1:
                for k,v in batch.items():
                    if k in tensor_items:
                        batch[k]=torch.cat(batch[k],0)
                    elif k in seq_items:
                        batch[k]=torch.nn.utils.rnn.pad_sequence(batch[k],
batch_first=True)
                    else:
                        batch[k]=batch[k][0]
            else:
                for k in tensor_items:
                    batch[k]=torch.cat(batch[k],0)
                for k in seq_items:
                    batch[k]=torch.nn.utils.rnn.pad_sequence(batch[k],
batch_first=True)
            return batch

def _preprocessing(self,path,new_anns,vids,qstart):
                with open(path,"r") as rf:
                        anns=json.load(rf)
                qid=qstart
                translator = str.maketrans("", "", string.punctuation)
                for vid in anns.keys():
                        ann=anns[vid]
                        duration=ann["duration"]
                        descrip=" ".join(ann["sentences"])
                        for ts,q in zip(ann["timestamps"], ann["sentences"]):
                            file1 = open("simple_query.txt","a")
                            file1.write(str(qid)+"##"+q+"\n")

                            #with open("simple_query.json","w") as f:
                            #    json.dump({qid:q},f)

new_anns[str(qid)]={"timestamps":ts,"query":q,"descriptions":descrip,"tokens":utils.to
kenize(q.lower(),translator),"duration":duration,"video_id":vid}
                            qid += 1

                        vids.extend(list(anns.keys()))
                return new_anns, qid,  list(set(vids))
```

Appendix B Activity Net Hyperparameters

```
evaluation:
  evaluate_after: -1
  every_eval: 1
  print_every: 100
logging:
  print_level: DEBUG
  write_level: INFO
misc:
  dataset: anet
  debug: false
  exp_prefix: anet/tgn_lgi/LGI
  method_type: tgn_lgi
  num_workers: 4
  print_every: 100
  result_dir: results/anet/tgn_lgi/LGI
  tensorboard_dir: tensorboard/anet/tgn_lgi/LGI
  vis_every: 1
model:
  checkpoint_path: ''
  dqa_lambda: 0.2
  dqa_weight: 1.0
  glove_path: ''
  grounding_att_cand_dim: 512
  grounding_att_drop_prob: 0.0
  grounding_att_hdim: 256
  grounding_att_key_dim: 512
  grounding_hdim: 512
  grounding_idim: 512
  lgi_fusion_method: mul
  lgi_global_nl_drop_prob: 0.0
  lgi_global_nl_idim: 512
  lgi_global_nl_nheads: 1
  lgi_global_nl_odim: 512
  lgi_global_nl_use_bias: true
  lgi_global_nl_use_local_mask: false
  lgi_global_num_nl_block: 1
  lgi_global_satt_att_cand_dim: 512
  lgi_global_satt_att_edim: 512
  lgi_global_satt_att_hdim: 256
  lgi_global_satt_att_n: 1
  lgi_global_satt_att_use_embedding: true
  lgi_global_type: nl
  lgi_hp_hdim: 512
  lgi_hp_idim_1: 512
  lgi_hp_idim_2: 512
  lgi_local_do_downsample: false
  lgi_local_num_res_blocks: 1
  lgi_local_res_block_1d_hdim: 256
  lgi_local_res_block_1d_idim: 512
  lgi_local_res_block_1d_ksize: 15
  lgi_local_res_block_1d_odim: 512
  lgi_local_type: res_block
  loc_word_emb_vocab_size: 11125
  model_type: LGI
  num_semantic_entity: 5
  query_enc_emb_idim: 11125
  query_enc_emb_odim: 300
  query_enc_rnn_bidirectional: true
```

```
  query_enc_rnn_dropout: 0.5
  query_enc_rnn_hdim: 256
  query_enc_rnn_idim: 300
  query_enc_rnn_nlayer: 2
  query_enc_rnn_type: LSTM
  resume: false
  sqan_att_cand_dim: 512
  sqan_att_drop_prob: 0.0
  sqan_att_hdim: 256
  sqan_att_key_dim: 512
  sqan_qdim: 512
  tag_weight: 1.0
  use_distinct_query_attention_loss: true
  use_gpu: false
  use_temporal_attention_guidance_loss: true
  use_video_encoder: false
  video_enc_pemb_idim: 128
  video_enc_pemb_odim: 512
  video_enc_use_position: true
  video_enc_vemb_idim: 500
  video_enc_vemb_odim: 512
optimize:
  decay_factor: 0.5
  decay_step: -1
  init_lr: 0.00004
  num_step: 500
  optimizer_type: Adam
  scheduler_type: ''
test_loader:
  annotation_path:
  -  /home/jjw6188/vpmt-master/data/anet/captions/val_1.json
  -  /home/jjw6188/vpmt-master/data/anet/captions/val_2.json
  batch_size: 100
  data_dir: data/anet
  dataset: anet
  feature_type: C3D
  in_memory: true
  max_length: 25
  num_segment: 128
  split: val
  video_feature_path:  /home/jjw6188/vpmt-master/data/anet/feats/sub_activitynet_v1-
3.c3d.hdf5
  word_frequency_threshold: 1
train_loader:
  annotation_path:  /home/jjw6188/vpmt-master/data/anet/captions/train.json
  batch_size: 100
  data_dir:  /home/jjw6188/vpmt-master/data/anet
  dataset: anet
  feature_type: C3D
  in_memory: true
  max_length: 25
  num_segment: 128
  split: train
  video_feature_path:  /home/jjw6188/vpmt-master/data/anet/feats/sub_activitynet_v1-
3.c3d.hdf5
  word_frequency_threshold: 1
```

Appendix C Activity Net NN Trainer File

```
from src.utils import utils, io_utils

#from src.model.LGI import LGI
import seq2seq
from neweric_mt import Encoder, Decoder
from pipeline_utils import *

from seq2seq.loss import NLLLoss
base_vocab = ['<PAD>', '<UNK>', '<SOS>', '<EOS>']
padding_idx = base_vocab.index('<PAD>')
sos_idx =4960
eos_idx =4961

class VPMT(nn.Module):
        """Pipeline, including models, optimizer, forward functions, update
        """
        def __init__(self, arg, dataset, optim_params_type="All"):
                super(VPMT, self).__init__()
                self.dataset = dataset # datset name
                self.args = arg
                self.optim_params_type = optim_params_type
                self.vocab_size = 11125
                self.arg = arg.lgi_arg # LGI model uses its own parameters
                #self.LGI_model = LGI(arg)
                #self.init_LGI()
                self.weight_loss = False # if using weighted loss

                self.encoder = Encoder(self.vocab_size, arg.word_dim,
arg.text_embed_size)

                #if self.arg.tie_weights: # Use same embedding layer for LGI and VSE
                #           self.encoder.src_embed =
self.LGI_model.query_enc.embedding

                self.use_attn = self.args.use_attn

                """ Advanced setting, if the translation is simplified version """
                self.simplified_trans = True

                """ using descriptions or not"""
                self.no_desp = True
                #self.device=torch.device("cuda:3" if torch.cuda.is_available() else
"cpu")

                self.device=torch.device("cpu")

                if self.simplified_trans:
                        if self.dataset == "anet":
                                self.train_ids, self.test_ids, self.idx_vocab,
self.vocab_idx  = init_trans("data/anet")
                                dec_vocab_size = len(self.idx_vocab)
                        else:
                                self.train_ids, self.test_ids, self.idx_vocab,
self.vocab_idx  = init_trans("")
                                dec_vocab_size = len(self.idx_vocab)
                        self.itow = itow = {int(k): v for k,v in
self.idx_vocab.items()}
                else:
```

```
                        dec_vocab_size = self.vocab_size
                        #self.itow = {v:k for k,v in test_D.wtoi.items()}
                self.decoder = Decoder(1000, hidden_size=arg.text_embed_size,
vocab_size=dec_vocab_size)

                #if self.args.cuda:
                 #       self.encoder.cuda()
                 #       self.decoder.cuda()

                #self.loss_fn = ComplexLoss()
                self.get_parameters()
                self.nllloss = NLLLoss()
                self.dec_criterion =
nn.CrossEntropyLoss(ignore_index=padding_idx).cuda()
                #self.vseloss = ContrastiveLoss()


        def forward(self, net_inps,
gts_queries,init_hidden,src_out,batch_size,target_len, prefix="", mode="Train"):
                # Encode
                #src_out, init_hidden = self.forward_vse_emb(net_inps['query_labels'])

                # Decode*******change to the Youtube version****** need to confirm
batch size and target_len********

                print("batch_size",batch_size)
                print("target_len",target_len)
                print("net_inps:",net_inps)
                #print("*****batch size is ",batch_size)
                #print("*****target len is ",target_len)
                #gts_queries = translate_gts(gts['qids'], self.train_ids,
self.test_ids, self.args.max_len)
                x=gts_queries[0]
                #outputs=torch.zeros(batch_size,target_len,self.vocab_size).to(device)
                outputs=torch.zeros(batch_size,target_len,self.vocab_size)

                for t in range(1,target_len):
                    output,last_hidden=self.decoder(x,init_hidden)
                    outputs[t]=output
                    best_guess=output.argmax(1)
                    x=best_quess

                return outputs
                '''if self.simplified_trans:
                        gts_queries = translate_gts(gts['qids'], self.train_ids,
self.test_ids, self.args.max_len)
                        self.scores = self.decoder(net_inps['query_labels'].long(),
gts_queries.long(), init_hidden, src_out, 10, teacher_forcing_ratio=0.2)
                        self.compute_loss_nll(self.scores, gts_queries.long())

                else:
                        self.scores = self.decoder(net_inps['query_labels'],
gts['query_labels'], init_hidden, src_out, 10, teacher_forcing_ratio=0.2)

                self.compute_loss_nll(scores, gts['query_labels'])
                return self.nllloss
'''

        def update(self):
                #If weare using both decoder and encoder parameters
                if self.optim_params_type == "All":
                        if self.optimizer == None:
```

```
                self.create_optimizer()
                self.optimizer.zero_grad()

        #Perform back propogation on loss and train neural network
        self.nllloss.backward()
        self.optimizer.step()
        #Zero out all tensors for next cycle
        self.optimizer.zero_grad()
else:
        #If no optimizer, create an optimizer and zero out tensors
        if self.optimizer == None:
                self.create_optimizer()
                self.enc_optimizer.zero_grad()
                self.dec_optimizer.zero_grad()

        self.nllloss.backward()

        self.enc_optimizer.step()
        self.dec_optimizer.step()

        self.enc_optimizer.zero_grad()
        self.dec_optimizer.zero_grad()
```

BIBLIOGRAPHY

[1] Lukin, S.M., Hobbs, R. and Voss, C.R., 2018. A pipeline for creative visual storytelling. *arXiv preprint arXiv:1807.08077*.

[2] Knight, K., Nenkova, A. and Rambow, O., 2016, June. Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*.

[3] Ranjay Krishna, Kenji Hata, Frederic Ren, Li Fei-Fei, and Juan Carlos Niebles. 2017. Dense-captioning events in videos. In *International Conference on Computer Vision (ICCV).*

[4] Rodriguez, C., Marrese-Taylor, E., Saleh, F.S., Li, H. and Gould, S., 2020. Proposal-free temporal moment localization of a natural-language query in video using guided attention. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision* (pp. 2464-2473).

[5] Jonghwan Mun, Minsu Cho, and Bohyung Han. 2020. Local-global video-text interactions for temporal grounding. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10810–10819.

[7] Kawato, M., 1990. Feedback-error-learning neural network for supervised motor learning. In *Advanced neural computers* (pp. 365-372). North-Holland.

[8] Gomi, H. and Kawato, M., 1993. Neural network control for a closed-loop system using feedback-error-learning. *Neural Networks*, *6*(7), pp.933-946.

[9] Xin Wang, Jiawei Wu, Junkun Chen, Lei Li, YuanFang Wang, and William Yang Wang. 2019b. Vatex: A large-scale, high-quality multilingual dataset for video-and-language research. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4581–4591.

[10] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 6000–6010.

[11] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *North American Association for Computational Linguistics (NAACL).*

[12] Chen Sun, Austin Myers, Carl Vondrick, Kevin Murphy, and Cordelia Schmid. 2019. Videobert: A joint model for video and language representation learning. *arXiv preprint arXiv:1904.01766.*

[13] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Ruslan Salakhutdinov, and Quoc V Le. 2019. Xlnet: Generalized autoregressive pretraining for language understanding. *arXiv preprint arXiv:1906.08237.*

[14] Nils Reimers and Iryna Gurevych. 2019. SentenceBERT: Sentence Embeddings using Siamese BERTNetworks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3982–3992, Hong Kong, China. Association for Computational Linguistics.

[15] Lu, J., Batra, D., Parikh, D. and Lee, S., 2019. Vilbert: Pretraining task-agnostic visiolinguistic representations for vision-and-language tasks. *Advances in neural information processing systems*, *32*.

[16] Li, L.H., Yatskar, M., Yin, D., Hsieh, C.J. and Chang, K.W., 2019. Visualbert: A simple and performant baseline for vision and language. *arXiv preprint arXiv:1908.03557*.

[17] Xie, S., Sun, C., Huang, J., Tu, Z. and Murphy, K., 2018. Rethinking spatiotemporal feature learning: Speed-accuracy trade-offs in video classification. In *Proceedings of the European conference on computer vision (ECCV)* (pp. 305-321).

[18] Miech, A., Zhukov, D., Alayrac, J.B., Tapaswi, M., Laptev, I. and Sivic, J., 2019. Howto100m: Learning a text-video embedding by watching hundred million narrated video clips. In *Proceedings of the IEEE/CVF International Conference on Computer Vision* (pp. 2630-2640).

[19] Carreira, J. and Zisserman, A., 2017. Quo vadis, action recognition? a new model and the kinetics dataset. In *proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 6299-6308).

[20] Mithun, N.C., Paul, S. and Roy-Chowdhury, A.K., 2019. Weakly supervised video moment retrieval from text queries. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (pp. 11592-11601).

[21] Liu, X., Deng, Z. and Yang, Y., 2019. Recent progress in semantic image segmentation. *Artificial Intelligence Review*, *52*(2), pp.1089-1106.

[22] Hirasawa, T., Yang, Z., Komachi, M. and Okazaki, N., 2020. Keyframe segmentation and positional encoding for video-guided machine translation challenge 2020. *arXiv preprint arXiv:2006.12799*.

[23] Chen, Y.W., Tsai, Y.H. and Yang, M.H., 2021. End-to-end multi-modal video temporal grounding. *Advances in Neural Information Processing Systems*, *34*.

Jason Wang

Education

The Pennsylvania State University, *Schreyer Honors College* — University Park, PA
Major: Computer Science — August 2019 - May 2022

Work & Leadership Experience

Nyansapo AI (Nittany AI Competition Winner) — University Park, PA
Software Engineer, Technical Lead — January 2020 - Present
- Developed an AI software usable by volunteers and NGOs in Kenya that provides digital assistance to teachers
- Implemented custom-made Azure Machine Learning speech-to-text models suitable to Kenyan children
- Built a RESTful API for both instructors and students using Node.js and MongoDB
- Placed 1st place out of 59 total teams in the 2020 Nittany AI Competition, receiving over $12,000 in funding

Philips — Pittsburgh, PA
Software Engineering Intern — May 2021 - August 2021
- Created a custom Gradle build-plugin in Groovy to increase the configurability of compiler warnings
- Designed and integrated Groovy build scripts into the GitLab CI/CD pipeline at Philips to mandate code analyzers
- Implemented SonarQube quality gates and Code Scene Hotspot analysis into TeamCity by adding new build steps
- Campaigned and presented program-wide code quality changes to over 200 Developers, Engineers, and Scrum Masters through 3, hour-long presentations

The Undergraduate Research Society (URS) — University Park, PA
President — September 2019 - May 2021
- Connected over 200 fellow undergraduate students with professors through presentations and workshops with the end goal of facilitating research opportunities
- Headed and planned bi-weekly meetings and events regarding upcoming events and ongoing research projects

Research Experience

Department of Computer Science, Penn State University — University Park, PA
Undergraduate Researcher — December 2020 - Present
Advisors: Prof. Rui Zhang and Dr. Yanjun Gao
- *EVOQUER: Enhancing Temporal Grounding with Video-Pivoted Back Query Generation* paper accepted at the ViGIL NAACL 2021 Workshop and submitted to EMNLP 2021
- Trained a custom neural network to maximize the accuracy of temporal grounding across multiple datasets
- Utilized high-performance computing nodes in the Roar supercomputer to run custom PyTorch Data Loaders

Plant Village Research Group — University Park, PA
Undergraduate Researcher — February 2020 - January 2021
- Optimized the speed and size of TensorFlow object detection models used in the Plant Village Nuru app by exploring optimizations within the TensorFlow 2 object-detection API

Awards/Scholarships

The Evan Pugh Scholar Senior Award — 2021 - 2022
- Awarded to seniors who are in the upper 0.5 percent of their respective classes

The President Walker Award — 2020 - 2021
- Awarded to those who earn a 4.0 (A) cumulative grade-point during their first semester

Ivan Saidel Computer Science Award — 2018 - 2019
- Awarded as a merit-based scholarship for Computer Science students

Technical Skills

Languages: Python, Java, C, C++, SQL, JavaScript, HTML, Groovy, Verilog, R
Frameworks & DevTools: PyTorch, TensorFlow, Node.js, MongoDB, Azure, Android Studio, Linux, Git, Bash, Gradle, Postman, Vivado
Courses: Artificial Intelligence, Data Structures & Algorithms, OOP, Operating Systems, Theory of Computation, Database Management Systems, Systems Programming, Regression Analysis, Time Series, Probability Theory
Language Skills: English (Native), Mandarin (Fluent), Spanish (Intermediate), Japanese (Beginner)