

CS7643 Spring 2023 Final Project: Tini-Tiny Image Captioner

Evan Burchard, Christian Smith, Alex Pan, Jose Penalver Bartolome
Georgia Institute of Technology

May 4, 2023

Abstract

Image captioning is an area of deep learning that has high potential utility and is receiving massive public interest. This is because image and natural language processing are exploding in popularity due to models like StableDiffusion [1] and ChatGPT [2]. However, these systems and image captioning models in general tend to be extremely computationally expensive to train and run due to their number of parameters. Thus, experimenting with and extending these models is impossible for the average person with consumer-grade hardware. We tackled this problem by creating a smaller and lightweight image captioning system that can be trained and implemented end-to-end on a single GPU. In this paper, we explain the motivation, methodology, results, and pitfalls of our ideas. Our code can be found in this GitHub Repository: https://github.com/penalverbj/CLIP_prefix_caption_fork.

1 Introduction

According to the National Federation of the Blind, the rate of Visual Disability in the US is 2.3% [3]. Among this population, many rely on assistive technologies such as screen readers to interpret images found online. Without image captions populating the alt text of images, screen readers have no way of interpreting images, creating poor user experiences and liability for site owners. Additionally, captioned images serve as a basis for other tasks involving categorization and recommendations.

Today, these tasks still rely heavily on human labor and the best performing machine learning based systems tend to be too large to be easily implemented by individuals and smaller organizations. Through our research into smaller transformer-based systems, we explored the possibilities and limits of reducing the resources required for automating image captions through deep learning. Specifically, we attempted to use small transformers with the goal of training an image captioning model on a single consumer GPU. In this paper, we introduce the “Tini-Tiny” image captioner: a model that can be trained and deployed end-to-end on a single consumer-grade PC. We trained our model using the

COCO [4] dataset consisting of image-caption pairs.

2 State of the Art/Related Work

In this section, we discuss the current state-of-the-art with respect to small image captioning models. In particular, we review current knowledge distillation practices and the resulting small vision and language transformers that are produced by following them.

2.1 Image Captioning Models

Many current models use an encoder-decoder framework to compress images into a latent space and then project those hidden-dimensions to a vocab space [5]. Specifically, they take a large pre-trained vision classifier such as Swin or ViT to encode images and an even larger GPT or BERT language model to decode the image embeddings [6, 7, 8, 9]. Current innovations in improving the performance of these large models are to include skip connections between vision and language modules to share contextual representations across modalities, comprehensive pre-training to improve accuracy, and unified loss functions to generalize training objectives and reduce training time [10, 11, 5].

2.2 Knowledge Distillation

Knowledge distillation refers to the practice of compressing a large “teacher” model into a smaller “student” model with similar behavior as the teacher. The aim is to create compressed student models that are much easier to deploy and implement while retaining much of the accuracy of their state-of-the-art teachers. This is done by utilizing a pre-trained teacher and it’s inferred labels of a dataset to train an untrained student with fewer parameters than the teacher. The objective function used for back-propagation is a linear combination of the loss of the student logits with respect to the teacher predictions and the true labels itself.

The efficacy of knowledge distillation has been improved upon in several key ways. Firstly, student performance can be enhanced by pre-training as demonstrated by Turc et al

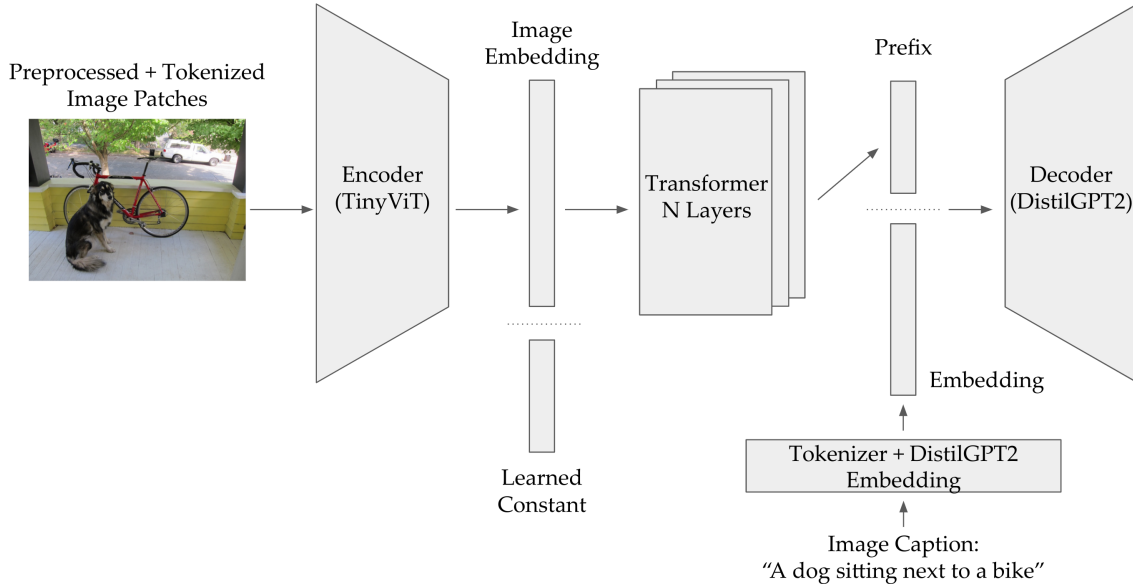


Figure 1: Training workflow and architecture of Tini-Tiny. Dotted-lines indicate vector concatenation. At inference, captions are generated only from the transformer produced prefix. See methods for specific implementation details.

[12]. Additionally, Sanh et al showed that including an additional student-teacher embedding loss term stabilizes training and improves student accuracy [9]. This methodology allowed for the creation of DistilGPT2, which has the same architecture as GPT2 but only contains 84M parameters as opposed to 124 million parameters in the original model.

Furthermore, Wu et al show that knowledge distillation training time and resources can be drastically reduced by simply running the teacher model’s forward pass on training samples prior to knowledge distillation [7]. This removes the forward pass bottleneck during training and frees up GPU memory. Wu et al use this to create TinyViT, a student with only 21M parameters compared to the 197M parameters of the Swin-L teacher [7, 6].

Despite these particular advancements, running zero-shot training for a student remained impossible on even a high-end consumer-grade PC. Wu et al report that teacher distillation still resulted in 3,360 GPU hours on an Nvidia V100 for the Imagenet-21k dataset. Mokday et al addresses this issue by introducing ClipCap, a training methodology that only learns a projection layer between the image encoder and language decoder. The fine-tuned weights of both the encoder and decoder are frozen. If fine-tuning is required, then decoder weights are also unfrozen. This drastically reduces the cost of training and so we model our methodology after Mokday et al’s. The novelty of our approach is that we replace the full CLIP and GPT2 models in ClipCap with TinyViT and DistilGPT2 to further decrease the training cost. We dub this the Tini-Tiny image captioning model.

3 Methodology

Figure 1 shows our Tini-Tiny image captioning model inspired by Mokday et al’s paper. Each section in the methodology describes the specific implementation of each module.

3.1 Data Selection

We chose to train our model using the full COCO 2014 dataset containing 82,783 training images and 40,504 validation images (each image has 5 different captions) for a total of 19 GB of disk space. COCO’s subject matter consists of photos of various objects in real-life scenes. It has a friendly API and a mature research community built around it. It should be noted that COCO does not adhere to the most up-to-date dataset standards as discussed by Gebru et al [13]. Additionally, COCO contains racial and gender biases [14].

3.2 Image Preprocessing and Encoder

First, input images were preprocessed in the same way as the original ViT implementation [7]. Briefly, images were resized to $1 \times 3 \times 224 \times 224$ and then segmented into 16 equally sized image patches. Each image patch was then combined with 2D-position embeddings and passed to pre-trained TinyViT convolution and transformer modules. Additionally, the head of TinyViT was removed since it was originally used for classification. The size of the embedded image then became $num_batches \times 576$. To reduce overhead during training, the embedded vector resulting from each training image was calculated and stored prior to training. Additionally, all trainable parameters were

frozen. It should be noted that TinyViT was fine-tuned on the Imagenet-22k dataset, not COCO.

3.3 Translation/Projection Layers

Transformers were used to map the image embeddings to vocab embeddings. To do this, image embeddings from the TinyViT encoder were concatenated with a learnable constant vector and passed into n standard transformer layers [15]. The constant vector serves to propagate meaningful image embedding information in each multi-head attention while also increasing the representational power of the image embedding itself. The resultant vector is referred to as a “prefix” vector by Mokday et al.

3.4 Decoder Training and Inference

The computation graph for training the decoder differs from performing inference using the decoder. The training loss on true captions is calculated auto-regressively. During training, the first j words in the true caption are embedded using the pre-trained DistilGPT2 word embedding layers. Then, the prefix vector is concatenated to the embedded caption. This concatenated vector is then passed through the rest of DistilGPT2 to predict the next word in the true caption. The training objective is to maximize the probability of the next word in the true caption given the prefix vector and the preceding words in the true caption. This was done by minimizing the cross-entropy loss:

$$L = \min_{\theta} \left(- \sum_{i=1}^N \sum_{j=1}^l \log p_{\theta}(c_j^i | p_1^i, \dots, p_k^i, c_1^i, \dots, c_{j-1}^i) \right) \quad (1)$$

where i is the index of a given image, j is the index of the true caption token, and $[p_1^i, \dots, p_k^i]$ and $[c_1^i, \dots, c_{j-1}^i]$ are the elements of the prefix and true caption token vector, respectively. The AdamW optimizer was used for gradient descent to optimize parameters θ .

During inference, only the prefix vector is passed to DistilGPT2 and each next word token is auto-regressively generated. A beam search (depth of 5) was used to choose the next best word.

3.5 Experimental Parameters

The same model parameters as Mokday et al were used unless otherwise specified. Importantly, the both the prefix vector size and batch training size were set to 40 for all experiments.

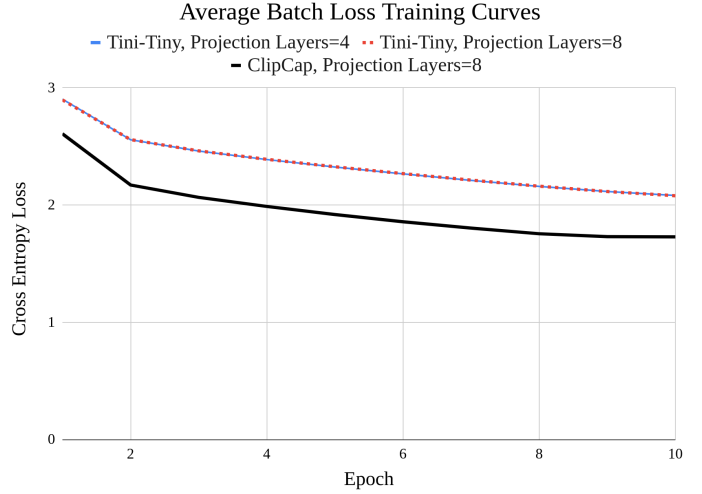


Figure 2: Training loss curves comparison between Tini-Tiny and ClipCap. It should be noted that Tini-Tiny with 4 and 8 projections have near perfect overlap.

4 Results

4.1 Tini-Tiny vs ClipCap

First, we assessed the performance of Tini-Tiny compared to the baseline ClipCap model, whose architecture has ~ 50 M more trainable parameters. Both models were trained on the same data for 10 epochs with a batch size of 40. Additionally, the number of transformer projection layers in each model was set to 8. Finally, decoder layers for both models were unfrozen for fine-tuning.

The first difference between Tini-Tiny and ClipCap was the training loss. Final average batch cross entropy loss after 10 epochs of training for the Tini-Tiny implementation was 2.08, and for the default ClipCap was 1.72. The training history shows remarkable similarity, as seen in Figure 2. Notably, this implementation of Tini-Tiny took 57 min/epoch to train on an Nvidia 2080Ti whereas ClipCap did not fit into GPU memory. Instead, ClipCap took 30 min/epoch to train on an Nvidia 3080Ti.

4.2 Relationship between projection layers and training loss

Next was to explore how the number of transformer projection layers between TinyViT and the DistilGPT2 in Tini-Tiny affected performance [15]. To assess this, we trained another Tini-Tiny model with only 4 projection layers. Further examining Figure 2, there was negligible difference in the cross entropy loss between the Tini-Tiny model with 8 projection layers compared to the one with 4. We conclude that there was enough representational power for learning to occur given the image embedding space and the prefix sizes that dictated the mapping network’s learnable parameter dimensions. However, this also indicates that our current

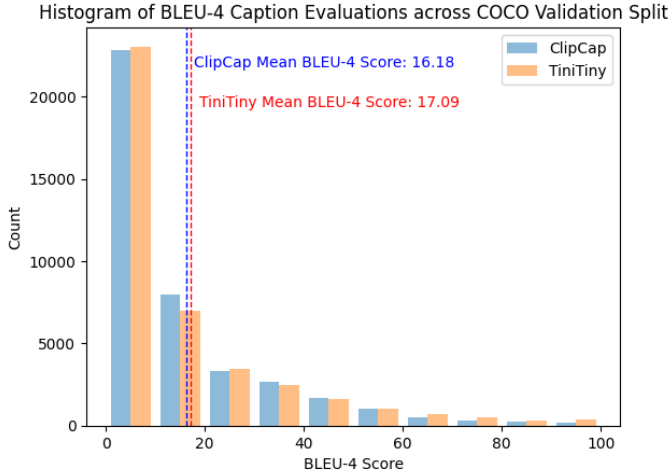


Figure 3: BLEU-4 evaluations of ClipCap and TiniTiny model generated captions when compared to ground truth captions.

implementations of Tini-Tiny are likely to overfit.

4.3 Performance on validation split

Finally, quantitative performance was evaluated via a BLEU-4 score on the COCO 2014 validation split [16]. Figure 3 shows the best BLEU-4 score achieved on each image in the validation split as a histogram. First, we were not able to reproduce ClipCap’s average BLEU-4 score of approximately 33. However, we suspect that this is because their reported model was also trained on the more comprehensive and larger Conceptual Captions data set before being evaluated on the COCO validation split.

With this context in mind, as seen in Figure 3, the Tini-Tiny model outperformed the ClipCap model with comparable hyperparameters when trained on the same data set for the same number of epochs. It is suspect that our smaller achieved better generalization than a larger model with higher representation power after training for the same number of epochs on the same data. Therefore, this adds to the conjecture that Tini-Tiny overfits the training and does not generalize well.

Examples of captioned images for both Tini-Tiny and ClipCap are shown in Figure 4.

5 Discussion

5.1 Conclusion

We show that using pre-trained student encoder and decoder models along with implementing a projection layer allows for end-to-end training and implementation on a single GPU. Additionally, these results show that simply learning the projection layers and fine-tuning the decoder provides sufficient

representational power to caption image embeddings optimized for a different dataset.

However, our image captioning showed inconsistent results. Our qualitative impression is that Tini-Tiny best captions images depicting simple outdoor and common indoor settings containing common objects and animals. This is likely due to the model having more exposure to these situations during training. When scenes were more specific or contained less common artifacts, our model’s captions were at times inaccurate in the context of the subject matter of the photo: Tini-Tiny often defaulted to the caption “A man riding a skateboard down the side of a ramp”.

We believe this happens for a couple of reasons. First, the TinyViT encoder was trained on Imagenet-21k, not COCO. Since the weights were frozen, it could have been identifying artifacts in COCO training data and facilitating the memorization of these artifacts instead of meaningful image features. Second, we believe our model overfits the training data and does not generalize well in part because only a single ground truth caption was used in training out of the five available. Third, it did not have enough exposure to more unique artifacts that rarely occur in the training data. It is also well known that the COCO dataset is biased on gender [14].

As described in the Future Direction section, we believe these shortcomings can be overcome with more experiments, training, and fine-tuning. Given this, we believe we were successful in creating a proof of concept model for our final project.

5.2 Initial Approach and Pitfalls

Before our final methodology for Tini-Tiny, we attempted to implement our own custom knowledge distillation inspired by the methodologies presented by Wu et al [7] and Sanh et al [9] instead. Namely, we first created teacher inferred labels prior to knowledge distillation and attempted to use a triple-loss objective to stabilize training. Our initial code base can be found in this GitHub Repository: https://github.com/gatech-edu/apan41/cs7643_image_to_text. The idea to make a tiny image captioning model always remained the same, but our approach in implementing it changed as we researched, planned, and attempted different strategies.

Parts of our plan that did not change were using COCO as our dataset and using TinyViT and DistillGPT2 as the encoder-decoder models. TinyViT was a natural fit for our goal due to its small number of trainable parameters and relatively high performance. Since TinyViT is a model that was specifically made for image captioning, we modified the source code to make it compatible with our goal. This included analyzing the inputs and outputs of different transformer and convolution modules and modifying their outputs to match up dimensions between encoder outputs and



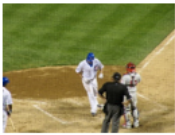

				
Image ID	COCO_val2014_000000000042	COCO_val2014_000000000133	COCO_val2014_000000000192	COCO_val2014_000000000359
Ground Truth	"A shoe rack with some shoes and a dog sleeping on them."	"A bed and a desk in a small room."	"A group of baseball players is crowded at the mound."	"A traffic light over a street surrounded by tall buildings."
Tini-Tiny	"A brown and white dog sitting on top of a couch."	"A bedroom with a bed, dresser and a window."	"A baseball player swinging a bat on top of a field."	"A man riding a skateboard down the side of a ramp."
ClipCap	"A close up of a person holding a pair of shoes"	"A picture of a trailer for a television show."	"A couple of men standing next to each other."	"A picture of a street with a lot of traffic."

Figure 4: Qualitative image captioning comparison.

decoder inputs.

After getting the pre-trained encoder and decoder matched up, we tried to implement a novel way of doing knowledge distillation. The first novelty was to NOT run the teacher and the students simultaneously during training, as described by Wu et al [7]. Because the forward pass of the teacher model takes longer than that of the student and is computationally expensive to hold in memory, training our model on a single device would have been impossible. Instead, we wanted to have all the teacher forward pass outputs computed and saved in a data frame prior to knowledge distillation so outputs could be queried during training time for loss calculation.

Our second novelty was to perform knowledge distillation using a triple loss calculated as follows: the teacher’s prediction logits with respect to student prediction logits, the student’s prediction logits with respect to the ground truth captions, and teacher’s prediction embeddings with respect to student prediction embeddings. Additionally, we attempted to freeze encoder and decoder layers that extracted high-level representations of the images and sentences. This was done to increase the performance of the encoder and decoder while decreasing training time.

However, storing both teacher embeddings and logits took up 500MB and 50GB of uncompressed disk space, respectively. While utilizing batch training would have reduced the total amount of GPU memory utilization, we realized our implementation would have been too resource-intensive for knowledge distillation on a single device to be feasible. We wanted to stay in scope and focused on the main goal of the project, so we pivoted to what became our final code base. We eventually found Mokady et al’s [17] approach to be more feasible, so we modified their novel approach to meet our goal.

5.3 Future Directions

There are several promising ways to extend our experiments and reduce Tini-Tiny’s overfitting issue. The first is to unfreeze the encoder weights so TinyViT can potentially learn more meaningful COCO image embeddings. Another solu-

tion is to employ additional loss terms inspired by Wu et al [7]. We opted to use a standard cross entropy loss in our experiments because it is simple to implement, but it would be interesting to assess how the addition of the KL divergence and cosine embedding loss would affect the results of our model. Additionally, we could test if adding skip connections between our projection layers improves the gradient flow and validation loss.

Finally, implementing knowledge distillation with a teacher image captioner and student Tini-Tiny as described by He et al [18] as a pre-training step could also improve model accuracy. Then, we could test how generalizable Tini-Tiny is by fine-tuning it on a different dataset such as Diffusiondb. Diffusiondb has a wider variety of different images and captions, so it would be a good test of our model’s ability to generalize spatial localization images and generate richer vocabulary usage.

6 Individual Contributions

Table 1 gives a summary of team member contributions.

References

- [1] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models, 2022. 1
- [2] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020. 1
- [3] W. Erickson, C. Lee, and S. von Schrader. Disability statistics from the american community survey (acs). ithaca, ny: Cornell university yang-tan institute (yti)., 2022. Re-

- trieved from Cornell University Disability Statistics website on 05.01.2023. [1](#)
- [4] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft coco: Common objects in context, 2015. [1](#)
 - [5] Jiahui Yu, Zirui Wang, Vijay Vasudevan, Legg Yeung, Mojtaba Seyedhosseini, and Yonghui Wu. Coca: Contrastive captioners are image-text foundation models, 2022. [1](#)
 - [6] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows, 2021. [1](#), [2](#)
 - [7] Kan Wu, Jinnian Zhang, Houwen Peng, Mengchen Liu, Bin Xiao, Jianlong Fu, and Lu Yuan. Tinyvit: Fast pretraining distillation for small vision transformers, 2022. [1](#), [2](#), [4](#), [5](#)
 - [8] Zijie J. Wang, Evan Montoya, David Munechika, Haoyang Yang, Benjamin Hoover, and Duen Horng Chau. Large-scale prompt gallery dataset for text-to-image generative models. *arXiv:2210.14896 [cs]*, 2022. [1](#)
 - [9] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter, 2020. [1](#), [2](#), [4](#)
 - [10] Chenliang Li, Haiyang Xu, Junfeng Tian, Wei Wang, Ming Yan, Bin Bi, Jiabo Ye, Hehong Chen, Guohai Xu, Zheng Cao, Ji Zhang, Songfang Huang, Fei Huang, Jingren Zhou, and Luo Si. mplug: Effective and efficient vision-language learning by cross-modal skip-connections, 2022. [1](#)
 - [11] Peng Wang, An Yang, Rui Men, Junyang Lin, Shuai Bai, Zhikang Li, Jianxin Ma, Chang Zhou, Jingren Zhou, and Hongxia Yang. Ofa: Unifying architectures, tasks, and modalities through a simple sequence-to-sequence learning framework, 2022. [1](#)
 - [12] Iulia Turc, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Well-read students learn better: On the importance of pre-training compact models, 2019. [2](#)
 - [13] Timnit Gebru, Jamie Morgenstern, Briana Vecchione, Jennifer Wortman Vaughan, Hanna Wallach, Hal Daumé III au2, and Kate Crawford. Datasheets for datasets, 2021. [2](#)
 - [14] Dora Zhao, Angelina Wang, and Olga Russakovsky. Understanding and evaluating racial biases in image captioning, 2021. [2](#), [4](#)
 - [15] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017. [3](#)
 - [16] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA, July 2002. Association for Computational Linguistics. [4](#)
 - [17] Ron Mokady, Amir Hertz, and Amit H. Bermano. Clipcap: Clip prefix for image captioning, 2021. [5](#)
 - [18] Ruifei He, Shuyang Sun, Jihan Yang, Song Bai, and Xiaojuan Qi. Knowledge distillation as efficient pre-training: Faster convergence, higher data-efficiency, and better transferability, 2022. [5](#)
 - [19] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale, 2021.

Student Name	Contributed Aspects	Details
Christian Smith	Implementation and Analysis	Research and architecture. COCO data preprocessing and training loop editing. Responsible for results section with their associated figures.
Alex Pan	Implementation, analysis, writing, and directions	Performed literature search, established to-do items and designed experiments, implemented and debugged encoder/decoder models, wrote related works, methods and (partially) results, created figures
Jose Penalver Bartolome	Environments, code, and writing	Responsible for setting up the conda environments through .yaml files. Involved in creating the COCO data loaders, and parsers, and plugging them into other parts of the code where they were necessary. Involved in generating the teacher output we would have used for knowledge distillation. Doing research for possible methodologies. Responsible for the Abstract and Discussion sections of the paper.
Evan Burchard	Research, architecture, implementation, and debugging	Researched and implemented state of the art architectures from relevant literature. Diagnosed and fixed functional and logical errors. Wrote Introduction, References, and some of Methodology.

Table 1: Contributions of team members.