AI Powered Ad Creatives Generation and Campaign

Christian Natajaya
Henry Samueli School of Engineering and Applied Science
Master of Engineering - Artificial Intelligence
University of California, Los Angeles
Los Angeles, CA 90024
chrisnat97@ucla.edu
UID: 805928302

Aaron Chen
Henry Samueli School of Engineering and Applied Science
Department of Computer Science
University of California, Los Angeles
Los Angeles, CA 90024
ac12@ucla.edu
UID: 005625945

Jeremy Szeto
Henry Samueli School of Engineering and Applied Science
Master of Engineering - Autonomous Systems
University of California, Los Angeles
Los Angeles, CA 90024
jeremyszeto@ucla.edu
UID: 505930009

Ryan Chien
Henry Samueli School of Engineering and Applied Science
Master of Engineering - Data Science
University of California, Los Angeles
Los Angeles, CA 90024
ryanchien@ucla.edu
UID: 905949417

Abstract

Generating creatives for online advertisements can be costly and inefficient. It is an even greater challenge to identify the creatives that lead to the optimal performance of an ad campaign. The typical process for this involves hiring a (human) agency to create multiple ad creatives for an advertisement campaign and manually running the selected creatives on the ad platforms (e.g. Facebook, Google) to find the best-performing ad. This can be very time-consuming and resource-intensive; the solution to this is to create a feedback loop on a AI image generator (e.g. DALLE-2) to create a self-improving ad creatives model that can programmatically generate advertisements resulting in the best performance.

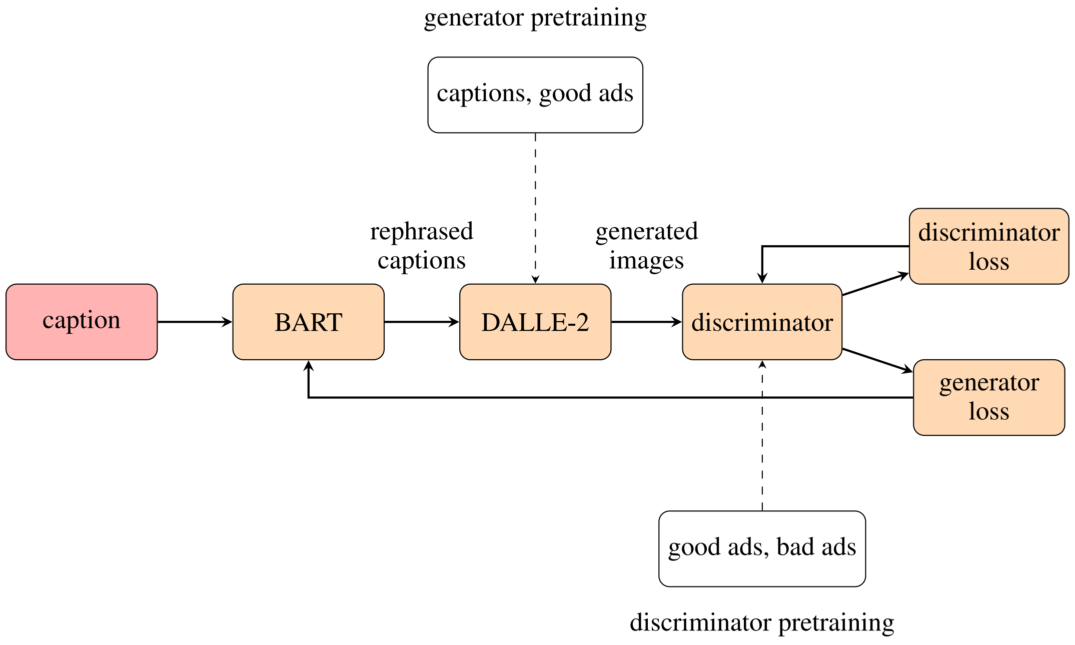

Figure 1: Model architecture

1 Introduction

Currently, DALLE-2 is only trained to create realistic images from text. The goal of this project is for DALLE-2 to create better ad creatives with a higher retention rate, not just to generate accurate images. In order to achieve this goal, we will implement a discriminator to improve the image generating process of DALLE-2. As illustrated in Figure 1, our model is composed of three main components: a BART (a denoising autoencoder for pretraining sequence-to-sequence models), a DALLE-2 (an AI system that takes a sentence as input and generates an image) and a discriminator that distinguishes between good ads and bad ads. The algorithm revolves around taking captions generated by BART, passing them into a pre-trained DALLE-2 model to generate an ad image, and utilizing the discriminator to adjust the weights on BART based on the image created to generate better captions to input into DALLE-2.

Our approach is novel and uniquely separates itself from currently existing approaches to this problem by incorporating a GAN loop to update the BART weights. Given an input (caption), the caption is passed through BART and eventually into DALLE-2. DALLE-2 then generates an image which the discriminator evaluates. The discriminator outputs a generator loss which is back-propagated to BART through the discriminator and DALLE-2 and is used to update the BART parameters. The discriminator and DALLE-2 both have their weights frozen so that when the generator loss is back-propogated through the entire model, only BART is effected by the back-propogation. This allows us to create a caption optimizer that learns to create captions that result in DALLE-2 outputting more successful ad creatives. The training process and algorithm is covered in more detail in the Section 3.

In this project, we will train the generator in two different ways:

1. Pre-train DALLE-2 using a set of ads and captions.
2. Use a GAN training loop to further train DALLE-2 based on the previous set of ad images.

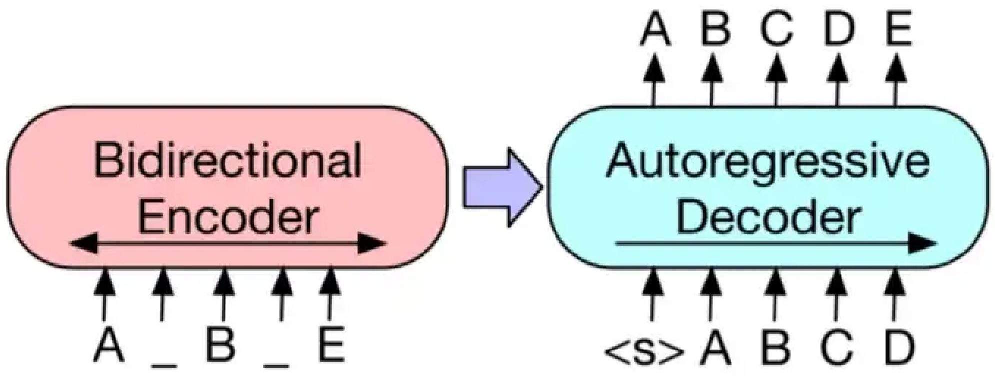

Figure 2: BART model flowchart

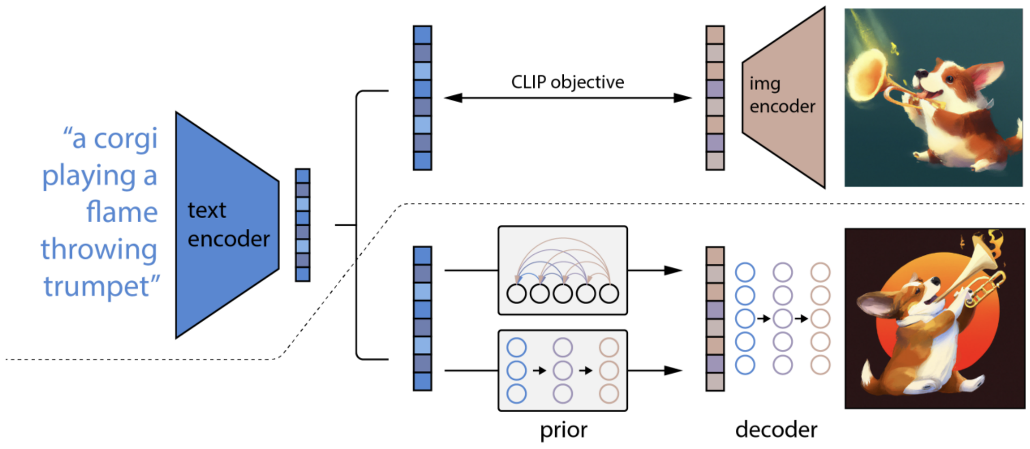

Figure 3: DALLE-2 model flowchart

2 Background

2.1 BART

BART (Bidirectional and Auto-Regressive Transformer) uses a standard sequence-to-sequence and machine translation architecture with a bidirectional encoder, similar to BERT (Bidirectional Encoder Representations from Transformers), and a left-to-right decoder, similar to GPT (Generative Pre-trained Transformer) as shown in Figure 2. BART is a denoising autoencoder for pretraining sequence-to-sequence models, such as text. BART is trained by (1) corrupting text with an arbitrary noising function, and (2) learning a model to reconstruct the original text. Pretraining BART involves randomly shuffling the order of the original sentences and corrupting the text with an abitrary noise function, where spans of text are replaced with a single mask token [1]. For the purposes of this paper, BART simply takes in a caption, encodes the caption, adds noise to that caption, and decodes it back into a new rephrased caption.

2.2 CLIP

CLIP (Contrastive Language–Image Pre-training) is a neural network (transformer) developed by OpenAI that is trained on a variety of image and text pairs as embeddings. The model pretrains an image encoder and text encoder by matching images with texts in the dataset, which is done by utilizing cosine-similarity. Now, the CLIP model is able to use this to make its predictions, even on unseen image classes ("zero-shot" predictions) [2, 3]. For the purposes of this paper, the image encoder is utilized during the DALLE-2 pretraining, and the text encoder is utilized during the forward pass of our DALLE-2 (see Figure 3).

2.3 Diffusion models

Diffusion models are generative models. Generative models try to generate data similar to what they are trained on. To do this, a diffusion model will add Gaussian noise to the data and then reverse the process by denoising. For the purposes of this paper, the diffusion model is used in DALLE-2 by taking in an image from pretraining and executing the noising/denoising process in order to produce a generated image [4].

2.4 GAN Loops

A GAN is a generative adversarial network, a deep learning network that takes in data as its input and outputs data with very similar characteristics. It consists of two main networks that work in tandem: a generator and a discriminator. The generator takes in input data and outputs data with similar characteristics, while the discriminator attempts to determine if the data is "real" or "generated". For the purposes of this paper, the generator is DALLE-2 and the discriminator is identifying whether the ad generated by DALLE-2 is a good ad (label=1) or a bad ad (label=0). This process is displayed in Figure 5.

3 Methods

All code for this project was run using a AWS P2 instance provided to our team by Professor Quan Quan Gu.

3.1 Dataset

The model requires a dataset that has a set of captions and a corresponding set of real ad images. We use an ad image dataset found on the University of Pittsburgh Department of Computer Science website [5]. The dataset contains 64,832 ad images that are verified by human annotators on Amazon Mechanical Turk. In order to generate the captions, we use an image captioning from Ankur Kumar [6]. The image captioning utilizes a Vision Encoder Decoder Model, which can be used to initialize an image-to-text model with any pre-trained Transformer-based vision model as the encoder (e.g. ViT, BEiT, DeiT, Swin) and any pre-trained language model as the decoder (e.g. RoBERTa, GPT2, BERT, DistilBERT). Image captioning is an example in which the encoder model is used to encode the image, after which an auto-regressive language model (i.e. the decoder model) generates the caption.

To distinguish between good and bad advertisement images, our team manually reviewed 700 images from the University of Pittsburgh ad dataset, labeling the image with a 1 if it was determined to be a good ad and a 0 if it was determined to be a bad ad [5]. Criteria we considered for good ads included strong brand recall - ads are more effective if the brand is clearly displayed in the creative - and a clear message - ads with one or a few large, simple graphics are more successful than those with multiple, complex features [7, 8, 9, 10]. A selection of the ads our team reviewed can be seen in Figure 4. Ads A, B, and D were assigned a 1 based on our criteria and ad C was assigned a 0.

Due to time and computational constraints, we were not able to train our model on the entire dataset. We decided to select 8 ads and captions to train our model on.

3.2 Training BART

We use a BART model to rephrase our captions. In the GAN training loop, we train BART instead of the DALLE-2 generator. We pretrain DALLE-2 to generate images that look like ad creatives, then we train our BART model in the GAN loop to produce the optimal captions for DALLE-2 to generate the best ad creatives. Our overall objective is to find the optimal captions in order to generate the best performing ads.

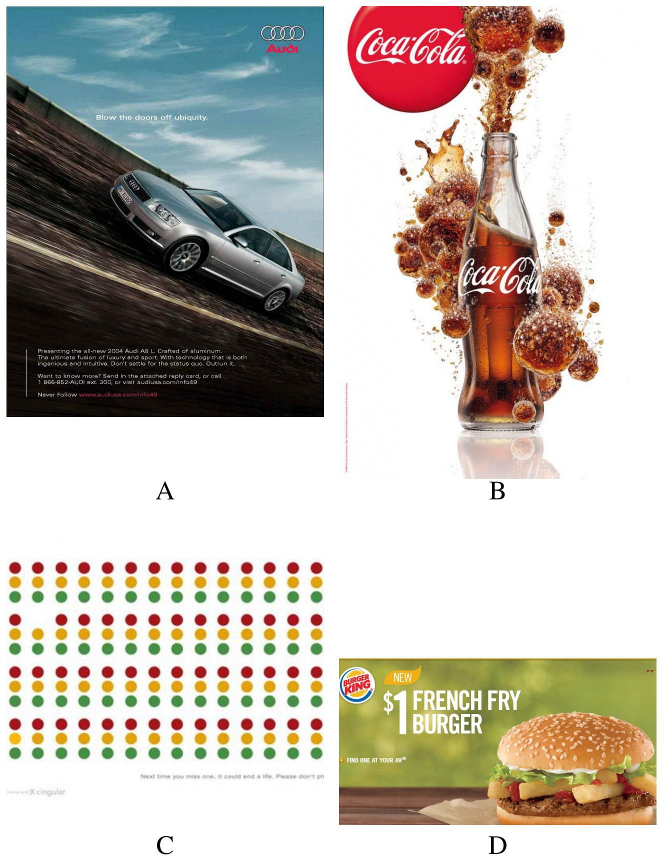

Figure 4: Sample ads from University of Pittsburgh ad dataset

3.3 Pretraining DALLE-2 generator

All code for the DALLE-2 model in our project was drawn from the lucidrains DALLE-2 repository on GitHub [11]. It contains the implementation of DALLE-2, OpenAI's updated text-to-image synthesis neural network, in Pytorch. Training DALLE-2 is a three-step process, with the training of CLIP being the most important. We use a pretrained OpenAI CLIP, imported from OpenAIClipAdapter [3]. Then, we train the decoder (a wrapping of CLIP and 2 U-Nets), which learns to generate images based on the image embedding coming from the trained CLIP. The diffusion prior network takes the CLIP text embeddings and tries to generate the CLIP image embeddings. Finally, both the decoder and diffusion prior network form DALLE-2. This is displayed in Figure 4. We take DALLE-2 and pretrain it on good ads and their corresponding captions.

3.4 Pretraining discriminator

The discriminator is a simple convolutional neural network, whose purpose is to be able to discern between good and bad ads. Therefore, we pretrain the discriminator on our dataset of selected images, with each image marked as 1 if the ad was deemed "good" and a 0 if the ad was deemed "bad" (reference Section 3.1 to see labeling process). When images from the generator are passed through the discriminator, we label these generated images as bad ads and evaluate them along with the training set of good and bad ads. It will then output both a discriminator loss and generator loss. The discriminator loss will be used to to improve its ability to discriminate between good and bad ads. In order to force the generator to continue improving its output, the loss is designed for the discriminator to adjust the weights such that the discriminator will eventually be able to completely differentiate between a good and bad ads. The generator loss will be mentioned in the following section about training the GAN loop (Section 3.5).

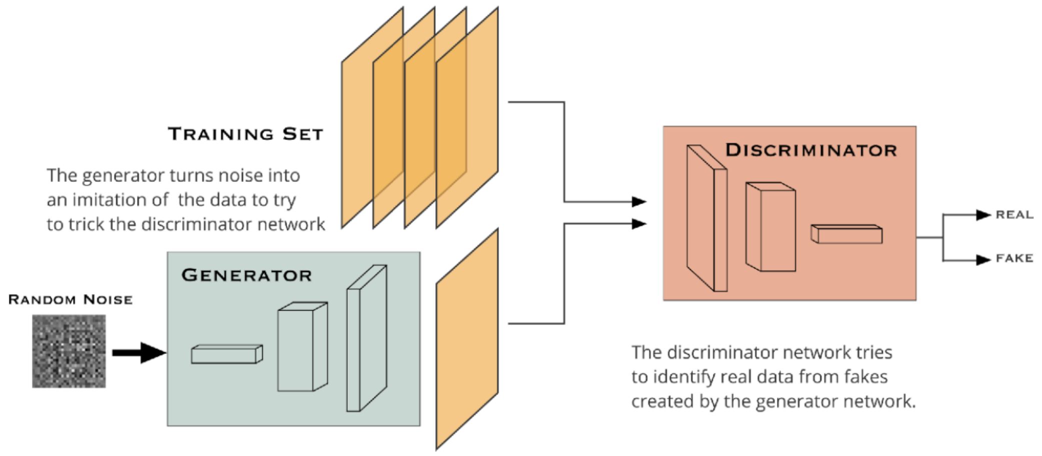
TRAINING SET
The generator turns noise into an imitation of the data to try to trick the discriminator network
RANDOM NOISE
GENERATOR
DISCRIMINATOR
REAL
FAKE
The discriminator network tries to identify real data from fakes created by the generator network.

Figure 5: GAN model flowchart

3.5 Training GAN loop

Our generator for GAN is DALLE-2, which receives text input (with brand and description, like "person sipping Coke bottle on the beach") and creates images. DALLE-2 uses a CLIP text encoder to create a text embedding, sends that through a diffusion prior network (which contains a causal transformer) to create an image embedding, then sends that to a decoder which is a diffusion model (built with U-Net architecture CNNs to denoise the image embedding).

The GAN loop is the feedback loop that back-propogates the generator loss from the discriminator and adjusts the weights in BART to generate better captions for DALLE-2. We froze the weights in DALLE-2 and the discriminator, since we did not want the output to be affected by the back-propogation of the generator loss. The readjusted BART weights generate a different rephrased caption to be passed into DALLE-2, and captions that correspond to bad images are passed into BART during the GAN training loop.

4 Evaluation and results

Our discriminator was successfully pretrained on our dataset of both good ads (label=1) and bad ads (label=0), as seen in Table 2. The DALLE-2 diffusion prior and decoder were also successfully pretrained on our dataset of good ads (label=1) and their corresponding captions, as seen in Table 1.

When training the GAN loop, we only passed captions corresponding to bad ads (label=1) into the generator, after which the generator creates an image and passes the image into in the discriminator to differentiate. During the first epoch of training, the discriminator was able to identify that the generator image outputs were bad, as can be seen by the high generator loss in Table 2. After the generator loss was back-propogated to BART, the generator learned to use these same captions to generate ads that were able to cause the discriminator into falsely classifying it as a good ad. This is significant because the discriminator had previously been pretrained on bad ads that correspond to these same captions. Now that the generator is able to use these same captions and fool the discriminator (see Figure 5), these ads generated by the generator *must* be better than the bad ads we used to pretrain the discriminator that correspond to the same captions. This also means that the GAN loop has effectively trained BART to assist DALLE-2 in making ad-creatives that are better than the bad ads in our dataset.

The GAN loop was very slow to train (six minutes per epoch for a training set of eight images corresponds with a training time of 10 hours for a full 100 epochs), because we attempted to make our model able to generalize to any ad type and used as input a wide range of diverse ad images, not just focused on one brand or product.

It is also important to note that the discriminator significantly outperformed the generator. This is likely due to the small training set (see Challenges section). Because the training set is small, the

Table 1: Pretraining loss for each network

Network Name	Pretraining Loss
Prior	0.0012
Unet1	0.0164
Unet2	0.0218
Discriminator	0.0001

Table 2: GAN loop discriminator and generator loss

Epoch	Discriminator Loss	Generator Loss
1	0.4510	99.9286
2	3.0232	5.1258
3	0.3084	74.9489
4	0.0505	99.9554
15	0.0934	91.3964
100	0.0001	99.9997

possible distribution of good ads (label=1) will always be extremely small. Because of this, any generated image will eventually never be classified as a good ad. As a result, the generator loss will be very high and the discriminator will outperform the generator.

5 Challenges and future work

The largest problem encountered in the development process of this model is the training time. Initially we had hoped to train the whole model on a general and large dataset. But during training, the time required to complete such a goal was long. For reference, having a training set of just 8 images took around 10 hours and a training set of 700 images would take several days to complete. Given the time constraints and computational resources available to us, having a training time longer than 10 hours was infeasible. This issue led to the discriminator always overfitting and significantly outperforming the generator. This is because with such a small training set of good and bad ads, the discriminator determines during the pretraining that the distribution of good ads is a small and slightly general distribution. Then during the training of the entire model, the discriminator is further told to shrink the distribution of good ads. Because the training set of good ads is so small, the discriminator can shrink this distribution very quickly (until it only wraps around the good ads), reducing the generality of the distribution completely. As a result, no matter how the input is manipulated, the generator could never create an ad that belongs to the good ad distribution that the discriminator has learned. Essentially, this means that anything the generator creates is almost always in the bad ads category. Since all the generator images are made to have the bad images label, the discriminator loss is near zero and the generator loss is always high.

In the future, we hope to be able to have the computational resources and time at our disposal to train the model on a the full dataset of 64,832 images. This makes the discriminator have a more general distribution of good ads and allow it to generalize to any different type of advertisement.

References

[1] BART. [Online]. Available:
`https://huggingface.co/docs/transformers/model_doc/bart`. [Accessed: 09-Dec-2022].

[2] OpenAI, “CLIP: Connecting text and images,” OpenAI, 21-Jun-2021. [Online]. Available: `https://openai.com/blog/clip/`. [Accessed: 09-Dec-2022].

[3] CLIP. [Online]. Available:
`https://huggingface.co/docs/transformers/model_doc/clip`. [Accessed: 09-Dec-2022].

[4] R. O'Connor, "Introduction to diffusion models for machine learning," News, Tutorials, AI Research, 08-Sep-2022. [Online]. Available: `https://www.assemblyai.com/blog/diffusion-models-for-machine-learning-introduction/`. [Accessed: 09-Dec-2022].

[5] "Automatic Understanding of Image and Video Advertisements," Automatic understanding of image and video advertisements. [Online]. Available: `https://people.cs.pitt.edu/~kovashka/ads/#image`. [Accessed: 09-Dec-2022].

[6] A. Kumar, "The illustrated image captioning using transformers," GitHub - Ankur Kumar, 21-Nov-2022. [Online]. Available: `https://ankur3107.github.io/blogs/the-illustrated-image-captioning-using-transformers/`. [Accessed: 09-Dec-2022].

[7] "Best practices for image ads," Facebook. [Online]. Available: `https://www.facebook.com/business/help/388369961318508?id=1240182842783684`. [Accessed: 09-Dec-2022].

[8] "What makes an effective ad?," Help.ads.microsoft.com. [Online]. Available: `https://help.ads.microsoft.com/#apex/ads/en/51015/0`. [Accessed: 09-Dec-2022].

[9] "Effective advertising makes people remember your name," Back to top. [Online]. Available: `https://www.wolterskluwer.com/en/expert-insights/effective-advertising-makes-people-remember-your-name`. [Accessed: 09-Dec-2022].

[10] "Online ads: Keep it simple," Business News Daily. [Online]. Available: `https://www.businessnewsdaily.com/8918-keep-online-ads-simple.html`. [Accessed: 09-Dec-2022].

[11] Lucidrains, "Lucidrains/dalle2-pytorch: Implementation of dall-E 2, OpenAI's updated text-to-image synthesis neural network, in Pytorch," GitHub. [Online]. Available: `https://github.com/lucidrains/DALLE2-pytorch`. [Accessed: 09-Dec-2022].

Checklist

The checklist follows the references. Please read the checklist guidelines carefully for information on how to answer these questions. For each question, change the default **[TODO]** to [Yes] , [No] , or [N/A] . You are strongly encouraged to include a **justification to your answer**, either by referencing the appropriate section of your paper or providing a brief inline description. For example:

- Did you include the license to the code and datasets? [Yes] See Section 5.
- Did you include the license to the code and datasets? [No] The code and the data are proprietary.
- Did you include the license to the code and datasets? [N/A]

Please do not modify the questions and only use the provided macros for your answers. Note that the Checklist section does not count towards the page limit. In your paper, please delete this instructions block and only keep the Checklist section heading above along with the questions/answers below.

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [No]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes]

    (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See A
    (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [No]
    (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See 3

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
    (a) If your work uses existing assets, did you cite the creators? [Yes]
    (b) Did you mention the license of the assets? [Yes]
    (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] See A
    (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [No]
    (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [No]
5. If you used crowdsourcing or conducted research with human subjects...
    (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
    (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
    (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A Appendix

The source code for this project has been submitted to BruinLearn and is additionally available at the following link: `https://drive.google.com/file/d/1dm6QBG5AlAaksJjUwT2zcIXNbQP1YaDn/view?usp=sharing`.

Slides for the final project presentation can be found here: `https://docs.google.com/presentation/d/1klL_ZKoTkRSynAsISXIauegRlnGOD42coZ33ufjLL1M/edit?usp=sharing`.