ALPHA-CLIP Data Science Spring Project

Ava Collier, Jamin Goo, Hannah Shaw, Bryant Willoughby

April 27, 2025

North Carolina State University

1 Background & Introduction

Recent advances in artificial intelligence have significantly enhanced the ability to extract meaningful information from images with broad applications. This project investigates using Contrastive Language Image Pre-training (CLIP) to perform image information extraction and editing tasks (Radford et al. (2021a)). The primary goal is to assess the reliability and effectiveness of generative AI systems in handling image-based data. This research is conducted in collaboration with North Carolina State University and Sandia National Laboratories to advance the understanding of trustworthy generative AI and its real-world applications.

CLIP is a state-of-the-art AI model developed by OpenAI that bridges the gap between computer vision and natural language processing. Unlike traditional image classification models, which rely on predefined categories, CLIP learns directly from natural language descriptions associated with images. Trained on a variety of datasets with image-text pairs, CLIP excels at zero-shot learning, enabling it to recognize and describe images without requiring additional labeled data for specific tasks (Desai and Johnson (2021)).

However, CLIP is not without limitations. Its reliance on large-scale Internet data means that it can inherit biases present in its training sources, leading to potential unintended outputs, especially when cultural, contextual, or ethical factors are at play (Radford et al., 2021). Although CLIP excels at recognizing common objects, it can be suboptimal with more abstract or systematic tasks, such as counting objects in an image (Radford et al. (2021b)). While CLIP has demonstrated strong generalization across diverse datasets, its reliance on global image embeddings can lead to challenges in tasks that require fine-grained visual understanding.

Alpha-CLIP is an enhanced version of the CLIP model that introduces an auxiliary alpha channel to allow for more precise region-based image understanding. Unlike standard CLIP, which processes images holistically, Alpha-CLIP incorporates an additional input channel that highlights specific regions of interest using masks (Liang et al. (2023)). These masks can be manually defined or generated automatically by models such as the Segment Anything Model (SAM), enabling Alpha-CLIP to focus on user-specified areas while preserving contextual information (Kirillov et al. (2023)). This approach allows the model to maintain the broad visual recognition capabilities of CLIP while improving its ability to handle fine-grained tasks that require selective attention to certain parts of an image (Sun et al. (2023a)).

Alpha-CLIP demonstrates superior performance in various downstream applications, particularly in zero-shot image classification and multimodal large language models (MLLMs). In image recognition tasks, when provided with foreground masks, Alpha-CLIP outperforms the original CLIP in zero-shot classification in datasets such as ImageNet-S, improving accuracy by focusing on the relevant parts of an image (Deng et al. (2009)). It also integrates with MLLMs such as LLaVA-1.5 and BLIP-2, enhancing visual question answer (VQA) and image captioning by reducing hallucinations and enabling more precise region-based descriptions (Liu et al. (2024); Li et al. (2023)). In particular, when tasked with describing an image, Alpha-CLIP ensures the generated captions are based on a highlighted area, improving relevance and reducing misinterpretations caused by distractions in complex scenes (Sun et al. (2023a)).

2 Problem Statement

We initially proposed solving the ReCAPTCHA problem. This is a security puzzle that tests whether a user on a web-application is human. The goal of this puzzle is to recognize an object of a certain category in a 3x3 grid of nine unique images. The user would select any images that contain the specified object.

In exploring this problem, we identified several limitations in both the CLIP and ALPHA-CLIP models. As detailed in the Appendix Section 7.4, and supported by data in Section 7.3.1, we ultimately found that neither model, in its current form, can effectively solve the ReCAPTCHA task. Due to the computational demands of ALPHA-CLIP (see Section 7.4.4), we chose to narrow our focus to the CLIP model. See Sections 4.1 and 7.3.2 for an overview of CLIP and ALPHA-CLIP's respective architectures.

Given these findings, we revised our goals. Rather than attempting to solve a specific security challenge, we shifted toward a broader evaluation of CLIP's performance across multiple datasets, with the aim of validating existing claims and identifying areas for improvement. This revised approach is outlined in Section 5.1, using the datasets introduced in Section 3. To address known weaknesses, we also experimented with retraining the existing CLIP model. These methods are described in Section 4.2, with results discussed in Section 5.2.

3 Data

As we revised the direction of our project, we moved away from our originally constructed dataset and instead selected a random subset of six publicly available image datasets to evaluate CLIP's performance. These datasets were chosen to include both datasets that CLIP was trained on and datasets that it had not encountered during training. Together, they vary widely in terms of image content, descriptive labeling, and variation within and between categories. This design enables us to assess both in-domain performance and the model's generalization capabilities across both images and labels.

Datasets CLIP was trained on:

1. **MS COCO** – The Common Objects in Context (COCO) dataset, is a benchmark dataset developed by Microsoft which contains 328,000 image-text pairings and 80 unique, single-word object categories (Samet et al. (2020)). MS COCO captures complex scenes featuring multiple objects per image, and while each image is associated with a single label for classification purposes, the presence of multiple salient objects introduces labeling ambiguity. It also encompasses the broadest variety of categories of images within a single set across all datasets tested. This allows for a clear demonstration of the effect of highly variable images without the obfuscation introduced by more complex labels. Since this dataset was part of CLIP's original training data, it establishes a reference for in-domain performance.

2. **Flickr30** – This benchmark dataset was developed for sentence-based image retrieval and contains approximately 30,000 images, each paired with five human-written captions (Jain (2020)). The images were selected from six distinct Flickr groups and represent a broad range of everyday scenes including humans, objects, animals, and landscapes. In our classification task, the descriptive captions are treated as labels, and CLIP is considered correct if

it matches any one of the five captions. This structure gives CLIP greater labeling flexibility but also increases the challenge due to the natural variability in human descriptions. The complex label structure probes the limitations of CLIP to process extraneous words and parse labels that describe multiple objects. This data set forces CLIP to manage high variability in labels and images simultaneously.

Datasets CLIP was not trained on:

3. **Stanford Cars** – Developed by the Stanford AI Lab, this dataset includes 16,185 images of 196 classes of cars. The general label structure includes the make, model, and year of each car (Krause et al. (2013)). It presents an example of a fine-grained classification challenge due to not only the high visual similarity between classes, but also the large amount of overlap across class labels. There is significant overlap in the make and and model of cars in the data set, forcing the model to identify slight deviations within the images of similar classes. CLIP's performance on this dataset helps us gauge its ability to distinguish subtle inter-class differences with both images and labels.

4. **Google ReCAPTCHA** – This dataset consists of approximately 12,000 images derived from Google's ReCAPTCHA V2 challenges, split evenly across 12 diverse object classes (Mazurov (2022)). These images are commonly used for real-world CAPTCHA tasks that ask users to select all images containing a specific item (e.g., "bicycles" or "traffic lights"). This dataset serves as a simple baseline for CLIP's ability to handle a small set of classes with uncomplicated labels and considerable variation across classes for data it has not previously encountered.

5. **Flowers 102** – This dataset comprises 102 flower categories commonly found in the United Kingdom, with approximately 7,000 images in total (Nilsback and Zisserman (2008)). Each class contains between 40 and 258 images, with noticeable intra-class variation and overlapping visual features across categories. As with the Cars dataset, CLIP's performance here reflects its ability to generalize fine-grained visual distinctions in a natural domain. However, this dataset provides more diversity across the class labels; this allows for direct examination of the fine-grained classification ability of CLIP without the interference of ambiguous labels.

6. **German Traffic Sign Recognition Benchmark (GTSRB)** – Originally introduced as part of a classification challenge at the International Joint Conference on Neural Networks (IJCNN) in 2011, the GTSRB includes around 50,000 images of 43 distinct traffic sign classes (Stallkamp et al. (2012)). The images vary in resolution, lighting, and occlusion. To a human observer, many signs are difficult to distinguish due to their granularity and low image quality, making this dataset particularly challenging for zero-shot learning. The labels are more complex than a single object, containing short descriptive phrases for each sign. There is also overlap in labeling across multiple classes (i.e. "Speed limit (20km/h)" and "Speed limit (120km/h)") requiring meticulous labeling. However, the images across these classes each have clear distinguishing features. This allows us to examine the effect of CLIP on identifying the subtle variations in labeling without the effect of similarity across the image classes.

By comparing CLIP's performance on datasets it was trained on (MS COCO and Flickr30) with its performance on a diverse set of unfamiliar domains (Stanford Cars, Google ReCAPTCHA, Flowers 102, and GTSRB), we aim to systematically explore both the model's strengths and its limitations.

4 Methods

4.1 CLIP

The CLIP model learns visual concepts from natural language supervision. It jointly trains an image and text encoder to predict the correct pairings of a batch of (image, text) training examples Radford et al. (2021a). At test time, the learned text encoder synthesizes a zero-shot linear classifier by embedding the names or descriptions of the target dataset's specified categories. The following subsections describe the pre-training and CLIP model as seen in Figure 1.

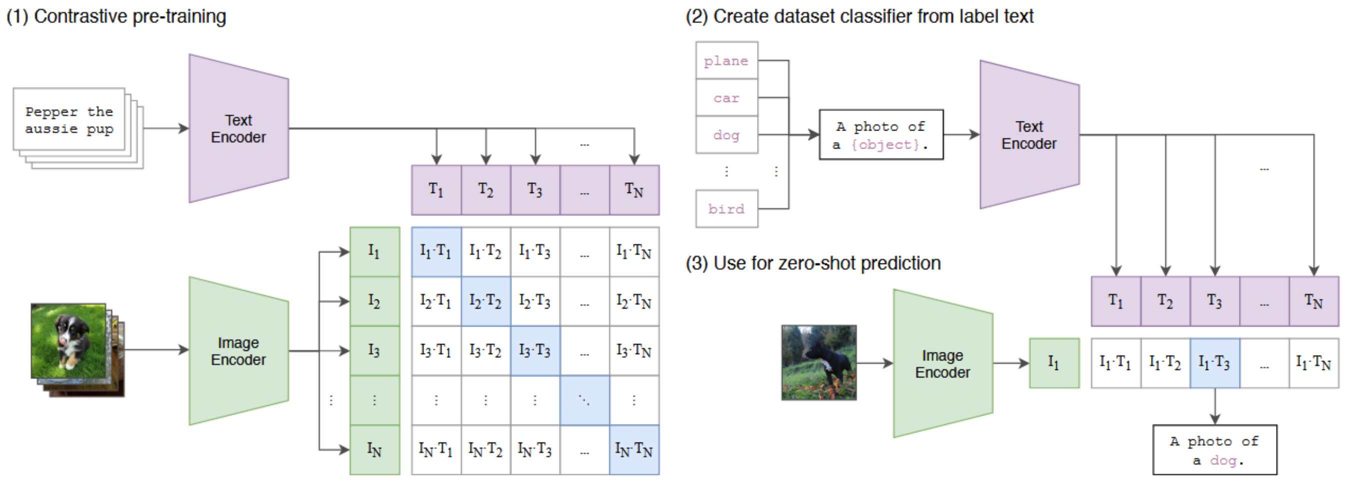


Figure 1: CLIP Model Framework

4.1.1 Contrastive Learning and Cosine Similarity

Contrastive learning is a machine learning technique that utilizes a measure of similarity instead of attempting to form a precise match. Similarity scores are calculated between images and text to form pairs, allowing for more flexible predictions, particularly when many similar options exist with very minor deviations. The CLIP model is trained using a measure of cosine similarity (1) between text and image embeddings, maximizing similarity for matching pairs. This is achieved by minimizing the angle between their respective vectors in the embedding space, a high-dimensional vector space where the encoded image and text data are mapped as numerical vectors. This assumes that each image is associated with a label when training. The labels must provide adequate contextual scope to classify all desired categories of images using the trained model.

$$c = \frac{\vec{a} \cdot \vec{b}}{||\vec{a}||\,||\vec{b}||} \tag{1}$$

where $\vec{a}$: encoded image vector and $\vec{b}$: encoded text vector.

4.1.2 Training with Multiclass N-Pair Loss

To utilize cosine similarity throughout the training process, an initial similarity score is calculated for each possible image-text pairing in the embedding space. Then, as the model is trained, multiclass N-pair loss (2) is performed in each iteration to update the model weights. This technique chooses one image as an "anchor" and then compares each potential similar text pair and *N* potential different text pairs (where *N* is the number of similar text pairs available), minimizing the distance between similar values and maximizing the distance between different values.

$$l = -\text{log}\left(\frac{e^{w(a\cdot p)}}{e^{w(a\cdot p)} + \sum_{i=1}^{N} e^{w(a\cdot n_i)}}\right) \tag{2}$$

where a: (anchor) image, p: positive (same class as anchor), n_i: negative (different class from anchor), and w: learned weight parameter.

4.1.3 Symmetric Cross Entropy Loss

After all iterations of training are complete, the similarity scores stored in the model weight variables are transformed into probabilities using symmetric cross entropy loss (5). This combines standard cross entropy loss (3), which forces convergence to improve predictions, and reverse cross entropy loss (4), which is more robust to noisy text labels. The output is a probability distribution over a fixed set of specified categories.

$$l_{ce} = -\sum_{i} Q(i) \cdot \text{log}(P(i)) \tag{3}$$

where $Q(i)$ is the true probability of each class and $P(i)$ is the predicted probability of each class.

$$l_{rce} = -\sum_{i} P(i) \cdot \text{log}(Q(i)) \tag{4}$$

where $Q(i)$ is the true probability of each class and $P(i)$ is the predicted probability of each class.

$$l_{sce} = \alpha l_{ce} + \beta l_{rce} \tag{5}$$

where α and β are hyperparameters controlling the balance between the standard and reverse cross entropy losses.

4.1.4 Optimization Techniques

The training process is further enhanced through various optimization techniques to ensure the model learns efficiently and reaches a state of numerical stability robust to noise. The Adam optimizer updates the image encoding model parameter weights, smoothing noise and increasing the learning rate. Adam is an adaptive, gradient-based optimization method that combines the ideas of Momentum and RMSprop to compute individual, adaptive learning rates for different parameters. Using an Adam optimizer allows the model to handle sparse gradients without eliminating certain classes and corrects biased gradients. The learning decay rate follows a cosine schedule, improving training speed and stabilizing the model. Additionally, the sharing process is used to find similarities across multiple GPUs, and gradient clipping prevents excessively large updates that could destabilize learning.

4.1.5 Zero-Shot Model and Deployment

The text encoder is a Transformer (Vaswani et al. (2023)), with the architecture modifications described in Radford et al. (Radford et al. (2019)). A transformer is a neural network architecture that uses an encoder-decoder structure, both of which are composed of identically stacked layers. Each layer contains two sub-layers: a multi-head self-attention mechanism and a simple, position-wise fully connected feed-forward network. The model was tested on various image encoders, with the base model provided using the recommended ViT-L/14@336px encoder.

CLIP is designed for immediate deployment. The pretrained model, hereby called the Zero-Shot Model, has been trained on over 400 million images and contains a dictionary of 49,152 tokens (words or phrases used to describe image contents). This allows the model to perform classification tasks without further training. The success of these tasks is explored in our analysis.

To utilize the CLIP model, the user simply provides an image and a library of descriptors (e.g., "A photo of $\{label\}$") that the image can then be matched with, ideally containing possible labels for the class of images being used. The model can also return single-word or phrase matches to the image if more detailed descriptions are not desired. The model then outputs the most similar label from the provided library.

4.2 Model Adaptation

While the Zero-Shot CLIP model demonstrates impressive performance, there is still room to improve the predictive ability; this can be accomplished through carefully retraining and adapting the architecture. Additionally, certain classes of images present challenges that hinder the CLIP model's predictive abilities such as: distortions and noise to images, similar but distinct class labels (i.e. Eastern White Pine and Western White Pine), images containing counts (i.e. 3 apples vs 4 apples), and any text contained within the images that offers distinguishing context. Utilizing additional data, transfer learning procedures, and minor changes to the architecture, we have thoroughly tested and constructed a retraining procedure to aid the CLIP model's fine tuning ability. This allows the model to be tailored to a more specific subset of images. Our retraining approach utilizes ensemble learning by harnessing the power of the Zero-Shot CLIP model's extensive training set, while building upon the context-specific details of the new subset and builds up a tolerance to noise and variability. Appendix Section 7.1 contains all of the files used for reproducing the subsequent retraining results.

4.2.1 The Underlying Structure of CLIP

To effectively address the issues that the Zero-Shot model faces, it is crucial to first understand the underlying architecture when training this model. As aforementioned, CLIP combines the output from a Vision Encoder and Text Encoder in an embedding space, subsequently using contrastive learning to find similarities. The encoders each output embeddings, which are then run through a projection head; a single linear layer that ensures proper formatting and dimensions. The resulting embeddings are then placed in a latent embedding space where similarity scores are calculated as described in Section 4.1. This workflow can be visualized in Figure 2. This architecture presents three primary locations to insert new data: the image embeddings, the text embeddings, and the embedding space. We explored each of these locations and addressed the results of each.

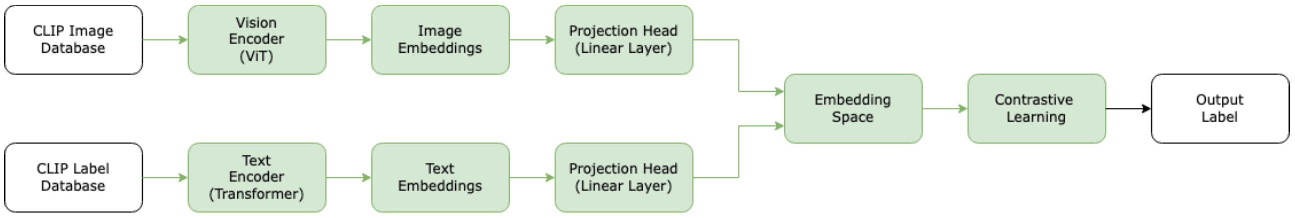


Figure 2: Base CLIP Training Workflow

4.2.2 Retraining

Retraining is the process of using the pre-trained weights from a model and continuing to train them with new data. Retraining allows for the powerful training in the Zero-Shot model across over 400 million images to be maintained while giving the model the opportunity to pick up the subtler differences specific to a smaller subset of data. We applied two approaches to retraining: fine-tuning and transfer learning.

In fine-tuning, no new architecture is added to the Zero-Shot model. This process simply continues to train the pretrained model weights using new data and the original pretrained architecture. This can involve freezing some of the layers, preventing certain layers of the architecture from being run on the new data. We tried each approach with freezing and without freezing in the CLIP model (as discussed further in Section 4.2.6). In general, retraining after normalization is the most numerically stable approach, however retraining the full model architecture can provide more notable improvements in performance.

Transfer learning is the process of retraining the weights of the pretrained model by freezing the original model architecture and adding an additional architecture head that is used to update the weights based on the new data. This introduces an alternative architecture compared to the base CLIP model. Additionally, transfer learning can be tailored a given data set, with architecture modifications informed by the data. Note that fine-tuning and transfer learning are not mutually exclusive; transfer learning can be applied in addition to fine tuning.

4.2.3 Ensemble Approaches

Ensemble methods involve constructing an entirely separate model that is trained in parallel and aggregating the predictions between the two models to balance the strengths and weaknesses of both. For CLIP, our new model is a classifier exclusively trained on the new data. The logits from both CLIP and this new model are then normalized and aggregated as a weighted average, which is then used to find the loss and guide the next iteration of training both models. The final prediction is found by minimizing the loss from the aggregate logits between an image and description. This allows the model to introduce a classifier without the noise of the pretrained weights interfering. This helps the model distinguish similar classes more accurately, refine generalization to unseen data, and drastically enhances the overhead cost of training the new architecture, as it finds distinctions much more rapidly.

The key to successful ensembling is finding the right balance between each model. The CLIP model identifies small details and relational-based features of an image well.(Radford et al. (2021a)). However, it is not robust to excess noise and can struggle to generalize to unseen data. Combining the new classifier with the original CLIP model helps to balance scope and allow a more robust fine-tuning procedure. Data sets with limited variation in labeling may need to rely more heavily on the classifier to tune out noise in the images and find clear distinguishing features. Data sets with limited variation in the images may need to rely more heavily on the CLIP model to find the small details that can be used for identification. For most data sets an even balance of both should be appropriate.

4.2.4 An Exploration of Model Architectures

To train CLIP on new data, we explored several architectures before we found one that was consistently successful in improving the accuracy of label predictions. In exploring, we gained important insights about the CLIP model performance. These include gaining additional understanding behind the challenges with certain dataset types and identifying components of the architecture with significant impact on the final performance.

To support informed decision-making, we divided the data into training, validation, and testing sets. This approach allowed us to evaluate potential overfitting, indicated by high accuracy on the training set coupled with declining accuracy on the validation set at each epoch of training; this discrepancy grows more extreme as the model is trained for further epochs when overfitting is observed. Given time and computational limitations, we did not use cross-validation.

First, we retrained both the vision and text encoder using transfer learning. We utilized a convolutional neural network (CNN) head for the vision encoder and a transformer head for the text encoder, each introduced at the projection head layer (as seen in Figure 3). This trained the Zero-Shot model weights on the new data. Note that in Figures 3, 4, and 5, the blue layers indicate where the pre-trained Zero-Shot model was frozen, the orange layers indicate where new data was introduced into the model and retraining or ensembling was conducted, and the green layers indicate where nothing from the CLIP training architecture was changed and training of the model was allowed to proceed as it would when training the Zero-Shot model.

We hypothesized that the convolutional layers in the vision encoder would focus on the importance of different spatial regions of the image. We also hoped the addition of the transformer head would enhance the text encoder in learning context-specific language patterns. However, this approach was unsuccessful. The transformer head offered no improvement in accuracy, regardless of tuning. The CNN head provided no significant changes in performance when less than five convolutional layers were used, and more than five convolutional layers resulted in a decrease in performance as a result of overfitting.

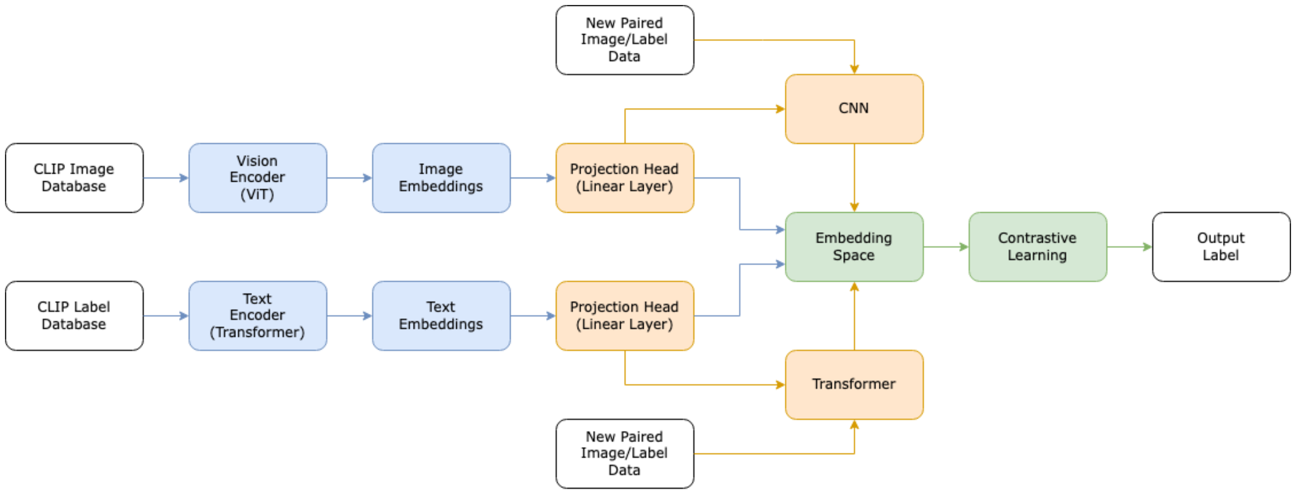


Figure 3: Transfer Learning Workflow
Blue: Frozen Layers from Zero-Shot Model
Orange: New Architecture Introduced
Green: Training with Zero-Shot Architecture

Next, we sought to fine-tune the vision encoder and introduce ensemble methods. Initially, we retrained the vision encoder using fine-tuning and a very low learning rate. We believed this constrained approach would allow us to modify the base CLIP model without overfitting. We refrained from retraining the text encoder as cursory experimentation indicated there was no notable benefit. We then trained a transformer classifier in parallel, and aggregated the normalized logits of both models, which were used to update the weights and form predictions. This architecture can be visualized in Figure 4. We chose a transformer with its complex architecture conducive to high classification accuracy, especially for images with minor deviations across classes. We hoped the combination of this architecture with the CLIP model could improve robustness and distinguish similar classes.

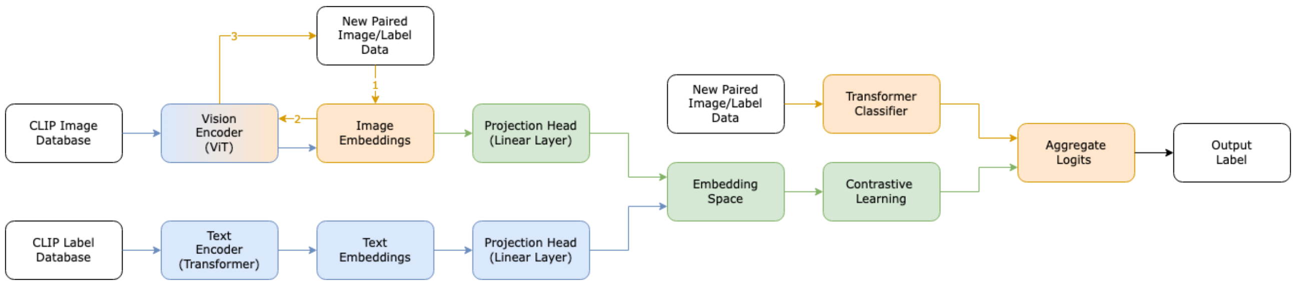


Figure 4: Fine-Tuning and Ensemble Transformer Classifier
Blue: Frozen Layers from Zero-Shot Model
Orange: New Architecture Introduced
Green: Training with Zero-Shot Architecture

This architecture introduced an unexpected obstacle: a drastic spike in overfitting. We experimented with a variety of training approaches to mitigate overfitting such as dropout, cosine and plateau schedulers, learning rates adjustments and weight decay, gradient clipping, and early stopping. While these approaches improved result, they could not eliminate the overarching issues.

Further investigation of logit weighting from the CLIP model and transformer classifier showed that both were severely overfitting, making this architecture incapable of generalizing to unseen data successfully.

Although this model was ultimately unsuccessful, it provided valuable information on how overfitting in the Zero-Shot model exacerbates difficulties with fine-tuned classification and generalization. This is an inherent characteristic of the CLIP model that cannot be simply removed. However, with this knowledge, we were able to design a new approach to adapting the Zero-Shot model that could compensate and improve the overall accuracy.

4.2.5 Training with Perceptron Classifier

We changed our architecture from fine-tuning with an ensemble transformer head to fine-tuning with an ensemble multi-layered perceptron (MLP) head (as seen in Figure 5). The simpler perceptron model is able to capture the low level features, thereby identifying the primary object in the image more accurately. This model also generalizes much better to new data. Using this in combination with CLIP allows the model to more accurately determine where to focus and removes much of the noise, still using CLIP to distinguish between difficult classes. With this approach, we saw immediate success.

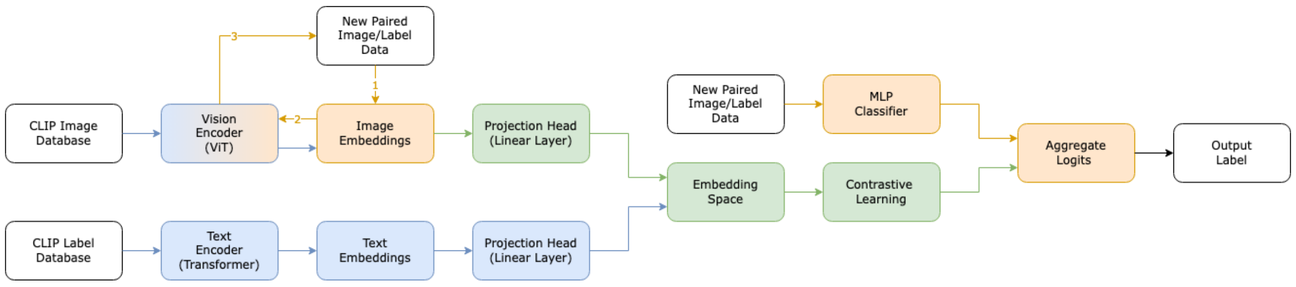


Figure 5: Fine-Tuning and Ensemble MLP Classifier
Blue: Frozen Layers from Zero-Shot Model
Orange: New Architecture Introduced
Green: Training with Zero-Shot Architecture

The vision encoder is still retrained using fine-tuning without freezing (as freezing notably dropped performance). The proposed MLP has a single linear hidden layer with 512 nodes, and is trained more aggressively to account for the fact that it has no prior information. The model uses an AdamW optimizer with weight decay and is placed on a cosine annealing scheduler; this forces the learning rate to decay on a cosine curve allowing for a high learning rate early on and a more moderate learning rate as the model begins to refine its weights. The logits of each model are normalized before aggregating and subsequently used to calculate cross-entropy loss to inform predictions. We then measured performance using accuracy on a test set, and compared to the performance on the Zero-Shot model.

4.2.6 Hyperparameter Tuning

The base version of this model was tuned extensively across five data sets: Stanford Cars (Krause et al. (2013)), Flowers 102 (Nilsback and Zisserman (2008)), Google ReCAPTCHA (Mazurov

(2022)), GTSRB (Stallkamp et al. (2012)) and Dogs (AmirMakir (2025)). We set the number of epochs to 10 but this can be increased as much as software constraints allow. We employ early stopping before reaching 10 epochs if the validation accuracy doesn't improve for three consecutive epochs (also known as a patience of 3). The other hyperparameters for this model include weight decay rate, batch size, hidden dimensions, learning rate of each model, learning rate scheduler, and aggregate weights. Each were optimized for general use (refer to Appendix Section 7.1), but for further fine tuning, we found the highest impact hyperparameters were the MLP learning rate and the aggregate weights. We set the MLP learning rate to $1e^{-4}$, which was suitable for most tasks. However, we found that for classes with similar labels or limited image variability across classes, increasing the learning rate could increase the final accuracy by 5-7%. Additionally, we found that if the model was struggling to distinguish similar classes, changing the weighting of the aggregate logit calculation could also improve the performance drastically. For the base model there is equal weighting; the specific model to weight more heavily was largely dependent on the type of data being used. Data that had very fine-grained differences among classes benefited from the high-level refined accuracy of the CLIP model. Such datasets could see better performance with a heavier weighting on the CLIP model. However, more diverse data classes that were overfit with the CLIP Zero-Shot model saw classification improvements with the perceptron classifier extracting low-level features; datasets of this nature might benefit from a heavier weighting of the classifier model. While the other hyperparameters can also be tuned, we did not identify any significant impact on performance.

5 Results

We evaluated CLIP's performance across several datasets by measuring Top-1, Top-3, and Top-5 accuracy in Section 5.1. These metrics indicate how often the correct label appears within CLIP's top 1, 3, or 5 predictions, respectively. To improve these accuracy scores, we implemented model adaptations for further training with the results provided in Section 5.2.

5.1 Performance Testing

As introduced in Section 3, two of these datasets (MS COCO and Flickr30) were part of CLIP's training data, while the remaining four were not. Our initial hypothesis was that CLIP would perform better on datasets it had seen during training. Table 1 presents the zero-shot accuracy scores across all datasets.

All images were preprocessed using CLIP's transformation architecture available at: Radford et al. (2021b), so that accurate comparisons can be made to the results seen by Sun et al. (2023b); this is for both evaluation and retraining. This includes resizing the images to 224×224 pixels, centering, and normalizing. The text was modified to fit the CLIP text encoder batch size of 77 tokens.

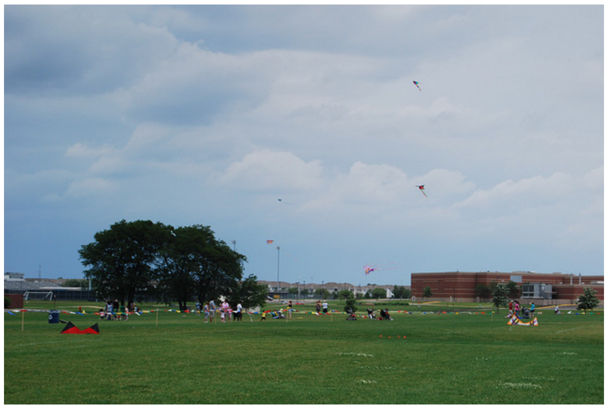


Figure 6: MS COCO image labeled as “person”

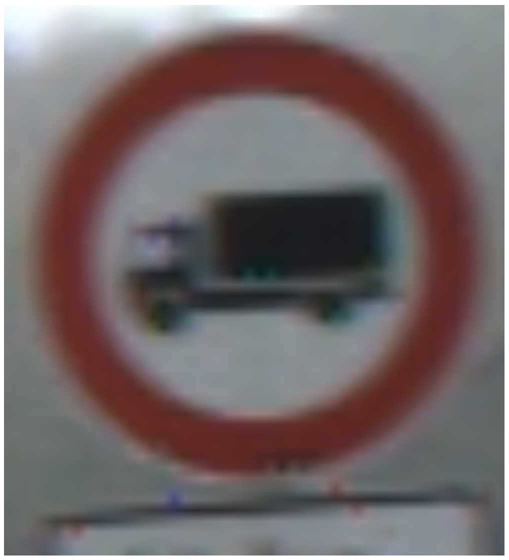


Figure 7: GTSRB image with low resolution

	Trained On		Not Trained On			
	MS COCO	Flickr30	Stanford Cars	Google Recaptcha	Flowers102	GTSRB
Top-1	36.9	27.7	58.9	59.4	63.5	28.6
Top-3	54.4	43.2	82.1	82.7	81.3	39.0
Top-5	64.1	51.1	90.0	92.5	85.0	44.2

Table 1: Zero-Shot Accuracy Scores

Contrary to our hypothesis, CLIP achieved lower average accuracy scores on the datasets it was trained on compared to those it had not seen during training. However, this comparison overlooks important differences in dataset composition. For instance, Flickr30 includes five detailed captions per image, each with rich sentence structure and semantic depth that includes multiple classes of images. Accurately matching images to such complex text embeddings requires a nuanced semantic understanding, which increases prediction difficulty.

MS COCO, while using simpler single-word labels, contains a relatively large and diverse set of 80 classes. Many of its images are also visually complex and could reasonably correspond to different labels. For example, Figure 6 shows an image labeled as “person.” While people are present, the image also includes distracting elements, which complicates the label assignment. These types of confounding factors limit the effectiveness of simple accuracy metrics and highlight the difficulty of interpreting performance in isolation. Additionally, as we saw in Section 4.2.5, the primary concern with the Zero-Shot model is overfitting. Practically, this means that the model will be more inclined to struggle identifying objects across diverse datasets that depend on lower-level features to distinguish classes as opposed to datasets that have less variability and rely on more fine-grained details. Also note that while the other data sets were used in their entirety, MS COCO used a training sample of size 50,000, validation sample of size 5,000, and test sample of size 10,000 due to its much larger size and our own software limitations.

The datasets CLIP was not trained on exhibit higher aggregate accuracy. However, the individual scores across these datasets vary. Stanford Cars, Google ReCAPTCHA, and Flowers102 all show relatively strong results. Notably, Google ReCAPTCHA achieves the highest Top-3 and Top-5 scores, while Flowers102 performs best in Top-1 accuracy. These results suggest that CLIP can effectively distinguish between visually similar images, such as car models or flower types, under the right conditions. ReCAPTCHA includes a smaller number of classes (12), contributing to consistent predictions and has several classes with similar features (i.e. motorcycles, buses, and cars). This demonstrates further evidence of the overfitting discussed in Sections 4.2.4 and 4.2.5 and reaffirms our training plans.

On the other hand, GTSRB yields the lowest performance across all accuracy levels. This dataset includes 43 traffic sign classes and contains many low-resolution or dark images (see Figure 7), which likely hinder CLIP's ability to identify fine-grained distinctions. These challenges underscore the limitations of zero-shot classification when image quality and class granularity vary significantly. Additionally, there is more natural variability across the image classes of this dataset that is overlooked in favor of the fine-grained details the Zero-Shot model, hence the need for a more simple perceptron classifier.

5.2 Retraining

Next, we present our results from our retrained model, using the base architecture outlined in Section 4.2.5, with model-specific fine-tuning governed by the hyperparameter configurations detailed in Section 4.2.6. This architecture aims to find the balance between fine-grain adaptation and new data generalization.

The updated accuracy outcomes of this retraining approach are presented in Table 2. Figure 8 shows the difference in accuracy scores after implementing the retraining approach. Note that accuracy scores for the Flickr30 dataset are not included. Unlike the other datasets, Flickr30 features complex, descriptive labels containing long descriptions that cover multiple classes. Additionally there is no overlap in the labeling across images. This complex structure introduced a current model limitation during retraining, particularly when attempting to evaluate accuracy on labels not seen during training. The model depends on the data being structured into classes that contain more than one image. This is to ensure that the model can be trained on all classes that it might encounter in a certain data category, and then tested on unseen data to ensure an accurate measure of generalized performance is captured. However, Flickr30 cannot be tested in this manner as there is no distinct class structure, and therefore it is not currently able to identify distinct classes in new images because the perceptron classifier accepts each label, in its entirety, as a class. Accordingly, we chose to omit these accuracy results for the time being, as they are not directly comparable to those from the other datasets. However, we evaluated the Zero-Shot performance metrics for Flickr30 (Table 1), as they provide valuable insight into CLIP's pre-trained zero-shot capabilities. In the future, we hope to experiment with the inclusion of a natural language processor to increase the usability of this model on more complex data structures.

	Trained On		**Not Trained On**			
	MS COCO	Flickr30	Stanford Cars	Google Recaptcha	Flowers102	GTSRB
Top-1	67.2	N/A	75.7	87.2	96.1	98.6
Top-3	80.8	N/A	93.0	97.4	99.4	99.6
Top-5	84.6	N/A	96.3	98.3	99.6	99.8

Table 2: Retrained Accuracy Scores

In all data sets, the predictive accuracy of the model showed a consistent improvement on the zero-shot baseline (see Figure 8). The most dramatic improvement was observed on the GTSRB dataset, where Top-1 accuracy increased from 28.6% to 98.6% - a jump of 70 percentage points. This suggests that the retrained model is not only capable of adapting to new data, but it is also significantly more robust to class imbalance and distortions. We believe the perceptron classifier was able to capture the low-level features of this data and build up a tolerance to noise. There was also an increase in Top-1 accuracy scores for the remaining datasets: Flowers102 (+32.6%), MS COCO (+30.3%), Google ReCAPTCHA (+27.8%), and Stanford Cars (+16.8%).

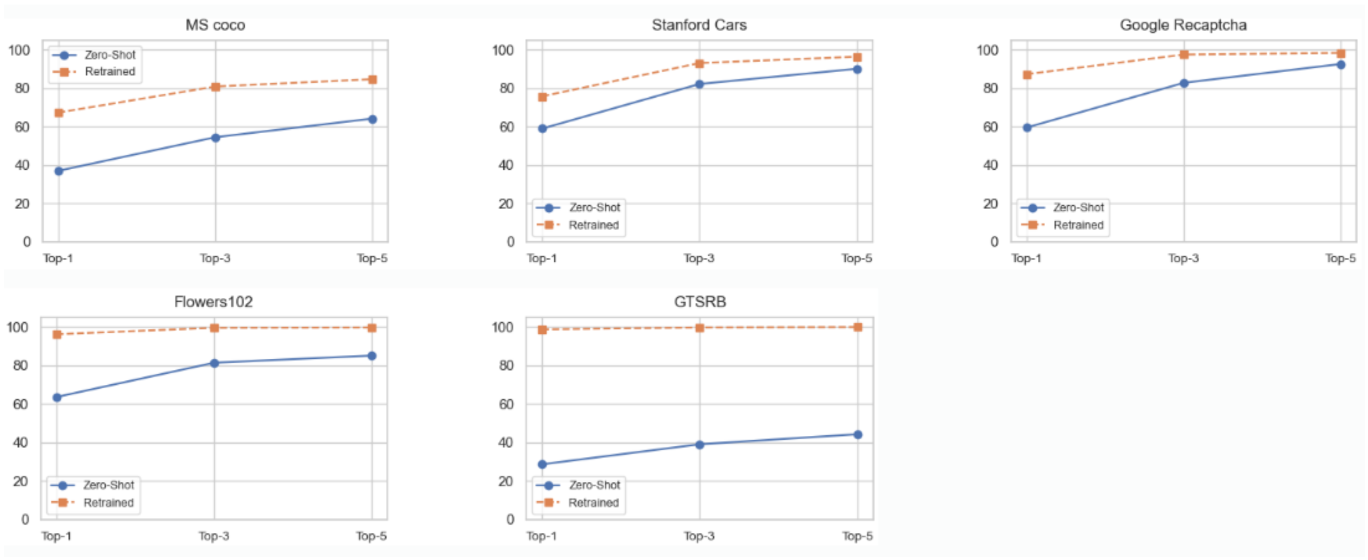


Figure 8: Accuracy Score Differences Before and After Retraining

We believe the variations in results across these data sets have a direct relationship to the structure, variability, and complexity of the images and labels. We saw the best performance in Flowers102 and GTSRB overall. Flowers102 had very distinct class labels and high quality images, directly focused on the flower of interest. This made it easier for the model to distinguish between, even very similar, classes. GTSRB also had labels with distinct differences in phrasing and length, as well as images that were clearly centered on the sign of interest, despite the distortions. Google Recaptcha and Stanford Cars also performed very well, however saw a notable drop in the top-1 accuracy compared to the top-3 and top-5 accuracy. For Google Recaptcha this can easily be explained by the more complex scenery in the images. The class label is not necessarily the primary object of focus in the image, and images can contain objects from multiple classes (i.e. a bicycle and a car could be in an image of a street scene). For this data set, a top-3 accuracy may actually be a more reasonable interpretation of the classes contained in the image. Stanford Cars was the first dataset where the features of the dataset started to challenge our model. Primarily, the similarities in the labels made it difficult to distinguish between similar year releases of cars, despite correctly interpreting the make and model. This is a dataset that would likely benefit from tuning the aggregation weight parameters to more heavily favor the CLIP model; this would allow the model to favor fine-tuned high-level features more than low-level features. The MS COCO dataset saw a notable improvement, but had the most diverse and complicated images across all datasets, with most images containing objects from at least two or three categories. This is reflected in the model performance, despite the simple label structure. This is an inherent characteristic of the dataset, and we believe we are nearing the saturation point of MS COCO using the CLIP model; additional tuning will likely have a limited impact on the accuracy because of the ingrained image complexity.

Overall, these results strongly support the proposed retraining strategy. There is evidence that the retrained model benefits from the low-level supervision of a perceptron model across unrelated tasks. The addition of the MLP classifier, the fine-tuned retraining of the Zero-Shot model, and the careful tuning of the hyperparameters not only boost fine-grained performance, but also improves the ability of the model to generalize to unseen data without overfitting.

The training process was fairly lightweight. Each data set was able to be trained on the adapted model architecture in Google Colab using the T4 GPU. The training process for a dataset of 50,000 is approximately two hours, while data sets containing 2,000 images can run in less than ten minutes. The model is designed for easy deployment and tuning to new data sets; it is quickly adaptable to multiple file structures. To utilize this model, the user must input a training set and validation set of images with corresponding labels.

6 Conclusion

Our investigation into CLIP's performance across both seen and unseen datasets has revealed a nuanced understanding of its capabilities and limitations. We initially hypothesized that CLIP would perform better on datasets present in its original training data, but our results demonstrated that the performance was much more dependent on the complexity of the labels and variation across the images. Zero-shot performance was highly dependent on the nature of each dataset and required limited in-class variability to perform well.

To address these limitations, we implemented a retraining and ensembling approach using an MLP classifier. This method led to consistent and incredible improvements in accuracy across all datasets, with some of the most significant gains seen in GTSRB and Flowers102. The retrained model not only improved Top-1 accuracy but also demonstrated enhanced robustness to domain-specific noise and label granularity.

Our retraining process remained computationally efficient, running on modest hardware within accessible timing constraints to make it easily deployable. This suggests that our framework could be a practical solution for adapting CLIP to new, classes with limited compute resources.

By expanding adaptability of CLIP through retraining, ensemble learning, and efficient architecture design, we hope to bridge the gap between generalized vision-language models and reliable, task-specific deployment.

References

AmirMakir (2025). Dogs dataset. Accessed: 2025-04-18.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255.

Desai, K. and Johnson, J. (2021). VirTex: Learning Visual Representations From Textual Annotations. pages 11162–11173.

Jain, A. (2020). Flick 30k dataset. Accessed: 2025-03-31.

Kirillov, A., Mintun, E., Ravi, N., Mao, H., Rolland, C., Gustafson, L., Xiao, T., Whitehead, S., Berg, A. C., Lo, W.-Y., Dollár, P., and Girshick, R. (2023). Segment anything.

Krause, J., Stark, M., and Deng, J. (2013). i l. fei-fei,3d object representations for fine-grained categorization ", u 4th international ieee workshop on 3d representation and recognition (3drr-13).

Li, J., Li, D., Savarese, S., and Hoi, S. (2023). Blip-2: Bootstrapping language-image pre-training with frozen image encoders and large language models.

Liang, F., Wu, B., Dai, X., Li, K., Zhao, Y., Zhang, H., Zhang, P., Vajda, P., and Marculescu, D. (2023). Open-vocabulary semantic segmentation with mask-adapted clip.

Liu, H., Li, C., Li, Y., and Lee, Y. J. (2024). Improved baselines with visual instruction tuning.

Mazurov, M. (2022). Google recaptcha image dataset.

Nilsback, M.-E. and Zisserman, A. (2008). Automated flower classification over a large number of classes. In *Indian Conference on Computer Vision, Graphics and Image Processing*.

Radford, A., Kim, J. W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., Krueger, G., and Sutskever, I. (2021a). Learning Transferable Visual Models From Natural Language Supervision. In *Proceedings of the 38th International Conference on Machine Learning*, pages 8748–8763. PMLR. ISSN: 2640-3498.

Radford, A., Sutskever, I., Kim, J. W., Krueger, G., and Agarwal, S. (2021b). Clip: Connecting text and images. Accessed: 2025-03-30.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. (2019). Language models are unsupervised multitask learners. *OpenAI*. Accessed: 2024-11-15.

Samet, N., Hicsonmez, S., and Akbas, E. (2020). Houghnet: Integrating near and long-range evidence for bottom-up object detection. In *European Conference on Computer Vision (ECCV)*.

Stallkamp, J., Schlipsing, M., Salmen, J., and Igel, C. (2012). Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition. *Neural Networks*, (0):–.

Sun, Z., Fang, Y., Wu, T., Zhang, P., Zang, Y., Kong, S., Xiong, Y., Lin, D., and Wang, J. (2023a). Alpha-clip: A clip model focusing on wherever you want.

Sun, Z., Fang, Y., Wu, T., Zhang, P., Zang, Y., Kong, S., Xiong, Y., Lin, D., and Wang, J. (2023b). Alpha-CLIP: A CLIP Model Focusing on Wherever You Want. arXiv:2312.03818 [cs].

Theron, G. M. (2023). Multi-label image captcha. Accessed: 2025-03-31.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2023). Attention is all you need.

7 Appendices

7.1 Appendix A.1: Retraining Files

The final model adaptation code and examples of performance on each data set can be found here.

7.2 Appendix A.2: Software Tools & Packages

Below is a list of required software tools and packages for the retraining architecture development and deployment.

1. Google Colab
2. Python 3.11
3. CLIP
4. CUDA 12
5. Python Packages:
   - i.) ftfy
   - ii.) regex
   - iii.) tqdm
   - iv.) Scipy
   - v.) Pytorch 2.6.0
   - vi.) torchvision
   - vii.) Pillow
   - viii.) os
   - ix.) NumPy
   - x.) Scikit-learn
   - xi.) shutil
   - xii.) packaging
   - xiii.) pandas
   - xiv.) kagglehub for importing the data

7.3 Appendix A.3: Supporting Work

Our initial project direction was largely unsuccessful, but it provided many interesting results. We describe our findings, results, and challenges here.

7.3.1 ReCAPTCHA

To explore the ReCAPTCHA problem, we recreated the grid-image structure using a multi-labeled dataset of 10,000 images from Google's ReCAPTCHA V2 Database (Theron (2023)). These RGB images are 100x100 pixels with a resolution of 72 pixels per inch, each assigned eleven binary labels corresponding to object categories such as bicycles, bridges, hydrants, and more. Since many images contained multiple objects, a multi-labeled dataset was necessary to ensure accurate classification.

To construct the grids, three images from a specified category were first selected, followed by six additional images sampled based on a probability density function (PDF). The PDF, as seen in Table 3, was designed to favor visually similar categories while maintaining variety. For instance, if the specified category was bicycle, the probability of selecting an image of a motorcycle was set at 0.25, while the probability of selecting a hydrant was only 0.01. This ensured that grids contained both similar and distinct categories. Each grid contains nine images, shuffled to ensure random placement, and saved as 312x312 pixel images with thin borders while preserving the original resolution and color structure.

	bicycle	bridge	bus	car	crosswalk	hydrant	motorcycle	stairs	tractor	traffic light	other
bicycle	0.25	0.01	0.1	0.1	0.01	0.01	0.25	0.01	0.2	0.01	0.05
bridge	0.01	0.25	0.1	0.02	0.25	0.02	0.02	0.25	0.01	0.02	0.05
bus	0.1	0.01	0.25	0.25	0.01	0.01	0.1	0.01	0.2	0.01	0.05
car	0.1	0.01	0.2	0.25	0.01	0.01	0.1	0.01	0.25	0.01	0.05
crosswalk	0.01	0.25	0.03	0.03	0.25	0.04	0.01	0.25	0.04	0.04	0.05
hydrant	0.04	0.04	0.1	0.1	0.04	0.25	0.04	0.04	0.15	0.1	0.1
motorcycle	0.25	0.01	0.1	0.1	0.01	0.01	0.25	0.01	0.2	0.01	0.05
stairs	0.01	0.25	0.03	0.03	0.25	0.04	0.01	0.25	0.04	0.04	0.05
tractor	0.1	0.01	0.2	0.25	0.01	0.01	0.1	0.01	0.25	0.01	0.05
traffic light	0.02	0.02	0.1	0.1	0.02	0.2	0.02	0.02	0.1	0.25	0.15

Table 3: PDF for Image Selection

7.3.2 ALPHA-CLIP

ALPHA-CLIP enhances the standard CLIP model by introducing an additional alpha channel to its image encoder, enabling region-based image understanding. This alpha channel acts as a continuous masking feature, similar to a grayscale, helping the model focus on important features of an image while reducing noise from less relevant regions. The ALPHA-CLIP framework offers two main tasks: a data generation pipeline (7.3.2) and a fine-tuning pipeline (7.3.2) for image classification. These pipelines are illustrated in Figure 9 and are described in the following subsections.

Data Generation Pipeline

The data generation pipeline can be used to construct a data set by extracting masks from specific regions and the utilizing ALPHA-CLIP and the Large Language Model BLIP 2.0 to generate individual images with corresponding labels. This pipeline will not be utilized in this project.

Classification Pipeline

The classification pipeline introduces subtle structural modifications to the CLIP image encoder to incorporate the alpha channel while retaining the model's prior knowledge. In the original CLIP model, the first layer applies an RGB convolution to the image. ALPHA-CLIP extends this by introducing an additional Alpha convolution masking layer parallel to the RGB convolution layer. This layer processes the alpha channel, which acts as a masking feature to identify important image

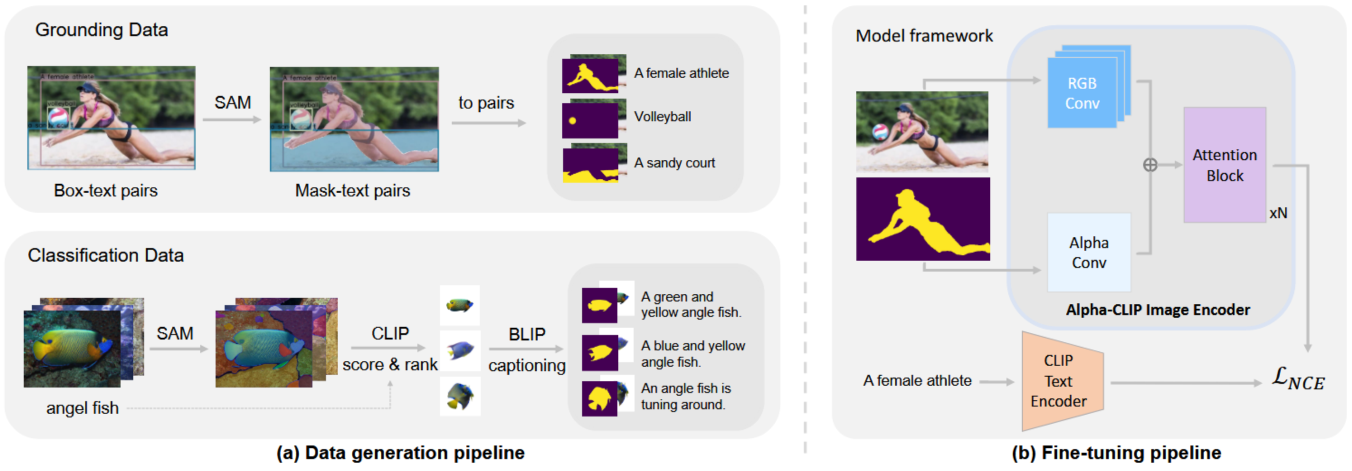


Figure 9: ALPHA-CLIP Model Framework

features. The outputs of the RGB and alpha channels are combined and passed through the same attention block as in the original CLIP model. This modification ensures that the alpha channel is seamlessly integrated into the model.

The alpha-channel input is normalized within the range [0,1], where 1 represents the foreground (region of interest) and 0 corresponds to the background (less relevant region). During training, the CLIP text encoder remains fixed, while the ALPHA-CLIP image encoder is fully trained. A lower learning rate is applied to the deeper transformer blocks to preserve CLIP's pretrained knowledge, while the first convolution layer (which processes the alpha channel) is assigned a higher learning rate to adapt quickly to the new input paradigm. A data sampling strategy is employed, where some RGBA-text pairs are replaced with original image-text pairs, setting the alpha channel to 1 to maintain compatibility with the original CLIP training data.

To facilitate learning, the Alpha convolution kernel weights are initialized to zero, ensuring that the model initially disregards the alpha channel and gradually learns its importance during training. The first convolution layer adapts at a faster rate, optimizing the use of the alpha channel, while the subsequent transformer blocks undergo more gradual fine-tuning. This training strategy ensures that ALPHA-CLIP retains CLIP's global recognition capabilities while smoothly adapting to the new alpha-based input paradigm.

The addition of the alpha channel alone does not actually improve the performance of the model, although it does maintain it. Instead, the relative benefits of using ALPHA-CLIP are seen in conjunction with the use of the the Segment Anything Model (SAM) to pre-process the images. SAM is pre-processing method that creates a large sample of masks of the image based on shapes and regions it distinguishes as highly correlated. Both CLIP and ALPHA-CLIP have been tested in conjunction with the SAM pre-processing method. While pre-processing with SAM increased the performance of both models, ALPHA-CLIP dominated in the prediction accuracy score with a two percent advantage. Utilizing SAM has other benefits as well, such as allowing the model to provide region level captioning and editing.

ALPHA-CLIP provides seamless integration with Large Language Models (LLMs). This can facilitate region-level captioning and answer questions about an image. The performance of these captions has been tested and made improvement in accuracy on other state-of-the-art models.

Using the ALPHA-CLIP Model

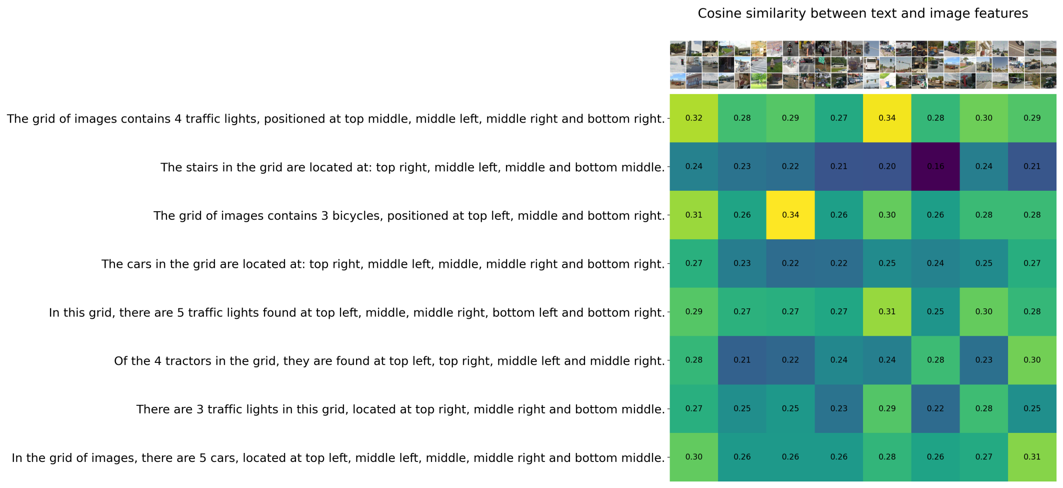
Running ALPHA-CLIP is generally the same process as running the original CLIP model. An image is given to the model as input. If desired, a SAM mask or another mask generated by the user can also be inputted in the model. There is still the option to provide labels and descriptors as well. The model will then output the most similar text description(s) to the image. If a mask was provided the text will primarily focus on the positive region specified by the mask.

Each grid was paired with corresponding textual labels to capture the desired information for the ReCAPTCHA task. These labels indicated the specified category, the count of images that contain objects in that category, and their positions within the grid. A prompt engineering approach generated structured sentence labels, such as: “There are {4} bicycles, located at {top middle, top right, top left, and middle}.” In total, 15 prompt variations were randomly assigned. Simplified labels, including just the category, count, and locations, were also created for testing purposes.

7.4 Solving ReCAPTCHA

7.4.1 Attempting CLIP

To approach solving the ReCAPTCHA problem, we first experimented with the CLIP model. To begin, we explored how the relationships and cosine similarity scores between the grid images and labels responded to this model. As seen in Figure 10, we already began to recognize two potential classification issues. Firstly, we began to see the similarity in the structure of the labels across different groups provided more noise that the baseline similarity calculations were able to filter out. Secondly, we saw that because the specified category is not required to be the most frequently occurring category of image in the grid, assessing the top-1 similarity score is not necessarily the best way to approach accuracy; a high similarity might not translate to an accurate description even more so than with standard images. To further examine CLIP’s ability to solve this problem we then ran the CLIP model on our grid data.



The grid of images contains 4 traffic lights, positioned at top middle, middle left, middle right and bottom right.	0.32	0.28	0.29	0.27	0.34	0.28	0.30	0.29
The stairs in the grid are located at: top right, middle left, middle and bottom middle.	0.24	0.23	0.22	0.21	0.20	0.16	0.24	0.21
The grid of images contains 3 bicycles, positioned at top left, middle and bottom right.	0.31	0.26	0.34	0.26	0.30	0.26	0.28	0.28
The cars in the grid are located at: top right, middle left, middle, middle right and bottom right.	0.27	0.23	0.22	0.22	0.25	0.24	0.25	0.27
In this grid, there are 5 traffic lights found at top left, middle, middle right, bottom left and bottom right.	0.29	0.27	0.27	0.27	0.31	0.25	0.30	0.28
Of the 4 tractors in the grid, they are found at top left, top right, middle left and middle right.	0.28	0.21	0.22	0.24	0.24	0.28	0.23	0.30
There are 3 traffic lights in this grid, located at top right, middle right and bottom middle.	0.27	0.25	0.25	0.23	0.29	0.22	0.28	0.25
In the grid of images, there are 5 cars, located at top left, middle left, middle, middle right and bottom middle.	0.30	0.26	0.26	0.26	0.28	0.26	0.27	0.31

Figure 10: CLIP Cosine Similarities for ReCAPTCHA Grids

Using the Zero-Shot Model, we input our grid data set of 10,000 grid images and their prompt-engineered labels (1,000 of each category excluding the *other* category). We did not provide additional labeling or descriptors. Using the prompt-engineered labels, we observed results approximately similar to randomly guessing; that is, top-1 accuracy 11.34% and top-3 accuracy of 31.78%. Note that top-1 accuracy refers to the highest probability text label that is output by the model being correct, and the top-3 accuracy refers to one of the three highest probability text labels being correct. As there are ten prediction categories, these results do not demonstrate any notable predictive power.

As these results were inconclusive, we then made a second attempt at using the CLIP model on our full grid data set. On this attempt time instead of using the prompt-engineered labels we simply used the single specified category labels (e.g. bicycle, bridge, etc.). Using this format, we observed a significant improvement in performance with a top-1 accuracy of 60.94% and top-3 accuracy of 93.75%. As previously stated, the top-1 accuracy is not necessarily reflective of how successfully the model is predicting the correct label because the specified category will not always be the most frequently occurring category of image in the grid. A superior criteria is the top-3 accuracy because at least one third of the images in the grid will *always* contain the specified category. Consequently, this time the model actually performed quite well in predicting the specified category of the grid. However, this does not take into account how many images containing the specified category exist in the grid or where they are located.

7.4.2 Limitations of CLIP

These analyses allowed us to identify several limitations of the CLIP model. The primary issue related to solving the ReCAPTCHA problem is that CLIP is designed for a single fine-tuned prediction. Consequently, it struggles to count and identify the location of an object in an image, even when it is able to correctly identify the specified category. This is not a problem that can be addressed simply by retraining; a segmentation process with iterative runs of the CLIP model on each segment would be required to allow each image to be evaluated individually. Additionally, the brevity of the text returned by the model is quite limited. While a variety of descriptions can be provided, the model only provides a single output; this means that even if the model was able to count or distinguish location there would not be an elegant solution to combining this information into a single description. Furthermore, the CLIP model (and consequently not the user) decides where attention is focused on an image. This makes the model simple to run and deploy on a large scale, but it also presents an issue for solving the ReCAPTCHA problem. If, for example, the specified category was bicycle and there was an image of a large bus with a much smaller bicycle next to it the model should recognize that the image contains a bicycle. However, because the bus is large it will attract more of the model's attention leading the image to be classified as bus instead, despite it containing a bicycle. This is an inherent constraint of the CLIP model that cannot be changed, presenting us will further difficulties when accessing accuracy.

That being said, CLIP was not without merits. It was very easy to deploy Google Colab, and despite limited GPU access it ran in a very reasonable amount of time given the size of the data set. It took approximately 30 minutes to complete both top-1 and top-3 labeling on all 10,000 312x312 pixel images (recall that it must compute the similarity between all potential labels images for every image in the data set). It also performed well on the classification task when given a single label input despite the excessive noise in the grid images.

7.4.3 Attempting ALPHA-CLIP

CLIP was clearly incapable of solving the ReCAPTCHA problem, so we then moved on to attempting this puzzle with the ALPHA-CLIP model. ALPHA-CLIP has several improvements that we believed would be advantageous to identifying individual images on a grid. Firstly, the use of masks would allow us to specify regional focus, presenting us with a built-in way to assess each image in the grid individually. Secondly, the use of SAM masks could aid in identifying features of the individual images that were not necessarily the most prominent part of the image as a whole. Thirdly, the use of SAM masks has been shown to further increase prediction accuracy over the CLIP model. Lastly, we believed this more individualized approach to classification could help us construct more detailed labeling with the use of a LLM because it can consider a specific region while retaining ”knowledge” of the entire context of the full image.

To begin we attempted running the Zero-Shot model on a single grid image using its specified category label, along with simple set of masks to identify each location on the grid. This did not perform as anticipated offering only an 11.11% accuracy (i.e. only one of the nine predictions was correct). However, we believed additional masking would be able to further improve the performance and that connecting the model to an LLM would allow us to create context-driven labeling. There are two recommended LLM models to accompany ALPHA-CLIP: LLAVA 1.5 and BLIP 2.0, both of which are adept at captioning tasks.

7.4.4 ALPHA-CLIP with an LLM

Unfortunately, we discovered that there were hardware and software constraints to running the ALPHA-CLIP model with an LLM. To provide the LLM with enough contextual information about the image outside of the specified region where a mask is applied, the ALPHA-CLIP model must be retrained with the LLM so that the LLM can be trained to recognize both region-specific and full image context. This means that to utilize an LLM in the way the model was designed, the entire ALPHA-CLIP model must be retrained. This is a computationally expensive task that requires hardware resources beyond those available in a standard laptop.

Several approaches were attempted to accomplish this retraining procedure. Firstly, we experimented with removing all GPU requirements, to test if it was even feasible to train the model on a basic laptop environment. We determined this is not an achievable task, not only because of complexity of the model, but also because both LLAVA 1.5 and BLIP 2.0 have CUDA integration within their required packages that could not be removed. We then attempted to run this training task on the virtual computer: Ubuntu 20 GPU with Cuda (GeForce RTX 2080 Ti). The training alone was, in fact, feasible in this environment, however there were timing constraints with our allowed access on this environment that made it impossible to install the necessary dependencies, train the model, and run the model within our allotment. This led us to explore the Hazel HPC environment through North Carolina State University. This environment demonstrated significant improvements on efficiency but had several other limitations we were unable to resolve. Firstly, we discovered that both LLAVA 1.5 and BLIP 2.0 require torch version 2.0.1. However, this version of torch was only compatible with CUDA version 11.7, which was not available on the HPC. Furthermore, both LLMs require significant disk space for installment, a minimum of 10 GB. We only had 1 GB of disk space allotted to this project making it impossible to install these packages correctly.

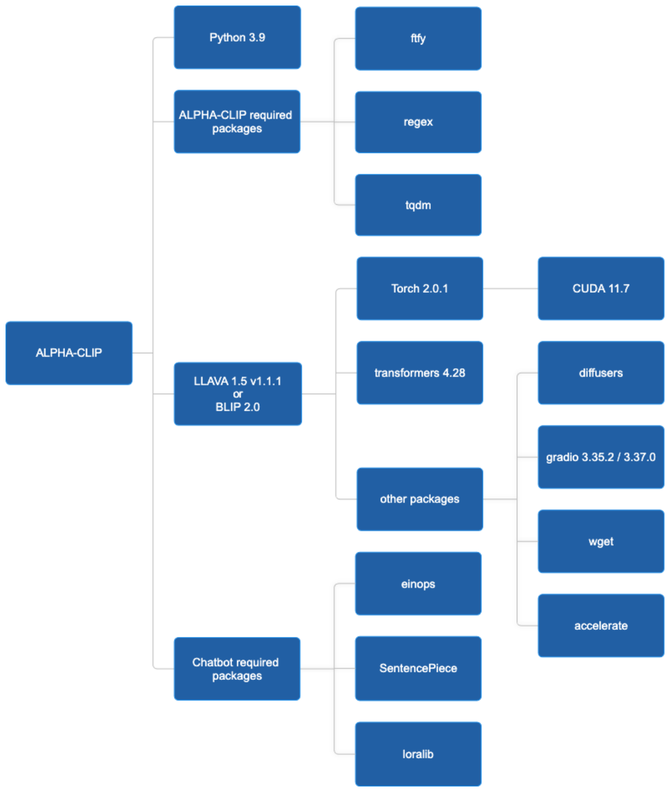


Figure 11: ALPHA-CLIP Dependency Chart

However, these challenges helped us develop a through understanding of the dependencies of the ALPHA-CLIP model, which are far more comprehensive than the CLIP model. Figure 11 demonstrates the dependencies of various aspects of the ALPHA-CLIP model that uses an LLM. Ultimately, these hardware and software requirements were unable to be resolved.

7.4.5 Limitations of Masking

Throughout the process of studying the requirement of running ALPHA-CLIP, we closely examined the underlying code of this model and noticed several constructs of the ALPHA-CLIP model the created additional barriers to solving the ReCAPTCHA problem. Firstly, discovered that only one mask can be applied to an image at a time. This is unfortunate because if we chose to apply a mask to recognize a specific image location in the grid, then we could not apply any addition masks; this meant we were unlikely to see any improvements to using the CLIP model with a segmentation algorithm. Secondly, there are functional constraints of using a mask that make it difficult to assess accuracy on a large scale. To obtain the improvements upon the CLIP model, in terms of predictive accuracy, a mask must be used. But, if a mask is used then the model generally requires some human input. In most cases, the location of an object to be identified in an image will not always remain in the same location across a variety images. This means that even if the masks are computer generated, a human must individually decide where to focus for each individual image. Consequently, it is hard to determine accuracy on a large scale unless using data with pre-made and labeled masks. In the context of ReCAPTCHA this meant that if we hoped to specify regions more specific than a general image's position on the grid it would require human input, which entirely defeats the purpose of this model solving the problem. Lastly, like CLIP, ALPHA-CLIP is only able to return single label output. This means that if a mask is used, the label produced will only correspond to the portion of the image the mask specifies. To get a comprehensive label of an entire grid ALPHA-CLIP would still need to be run iteratively on each image in the grid.

With all these constraints, we ultimately decided that neither model is capable of solving the ReCAPTCHA problem in their current states. Consequently, we decided to shift our goals to align with the, now discovered, constraints of these models.