Assignment 1

CSC2125H SE4ML, Fall 2023

Submitted by:	Jincheng Wang, Umar Masud
Student ID:	1004796385, 1010025857
UTorid:	wangj656, masuduma

Github link:

https://github.com/umar07/albumy

Changes can be viewed at
https://github.com/greyli/albumy/commit/bdb880beea89415b19a9923bd25cbac0ec5af956#diff-a488589b19dcbb8121c776b7e23b2b24825e16663968c357849c0f0c5e4dead3

Technical Description:

In order to make sense of the large codebase, we opted for the principle of divide and conquer. We took a top-down approach by deciding what we wanted to contribute or understand about the code base. We used break-points, logging and keyword searches to investigate how our top-level concerns flow down to the nitty gritties.

Once we had a clear idea of the code flow, we could easily make the necessary changes. Our task of implementing the ML-enabled features was centered around the event of uploading a new photograph. When a user uploads a new picture, we wanted to generate tags (that are searchable in the entire database) and also provide a description text which could act as the alternative text for the image.

Generating Automatic Tags:

To get the relevant tags for a new upload, we need to identify the salient objects in the image. For this purpose, we used an object detection model that takes the image as input and outputs classes/categories of the objects present in the image. These classes act as the relevant tags required. We use the YOLOv8 model from Ultralytics to get the object classes.

Code Description -
The files that required the changes were

- File = albumy/blueprints/main.py
  Function = upload() at line 125
  We call a new function get_tags(img) within this upload() function at line 129 to get a list of the tags relevant to the image.
  The get_tags(img) function is defined in a separate file that we created, named get_ml_tags.py . Here we import the ultralytics library, initialize the object detection model with pre-trained weights and perform inference. The output is a list of tags for the image.

  We then add the new tags at line 145 using the already defined Tag class. We make sure no same tags are added twice.

  These tags are automatically searchable by the original code design. So all the images with the same/similar tag will be shown in search results.

Generating Alternate Text:

The best way to get some description of the uploaded image is to simply use a pre-trained image captioning model. We do this with the BLIP model from Salesforce. The image is given as input to the model and we get a short description/caption (adjustable token size, kept to a maximum of 20) as the output, which is used as the alternate text. It is editable so can be overwritten by the user.

Code Description:

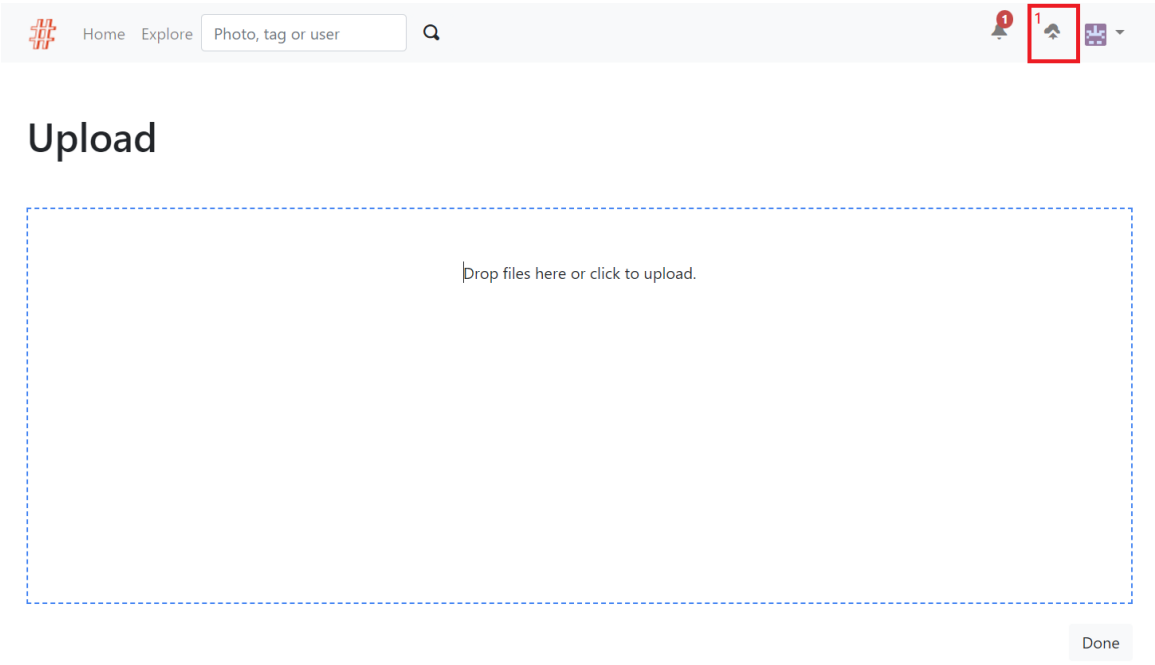
The files that required the changes were

- File = albumy/blueprints/main.py
  Function = upload() at line 125
  We call a new function caption(img) within this upload() function at line 130 to get a small description for the image.
  The caption(img) function is defined in a separate file that we created, named get_alternate_text.py . Here we import the BLIP code repo, initialize the image captioning model with pre-trained weights and perform inference. The output is a short, alternate text for the image.

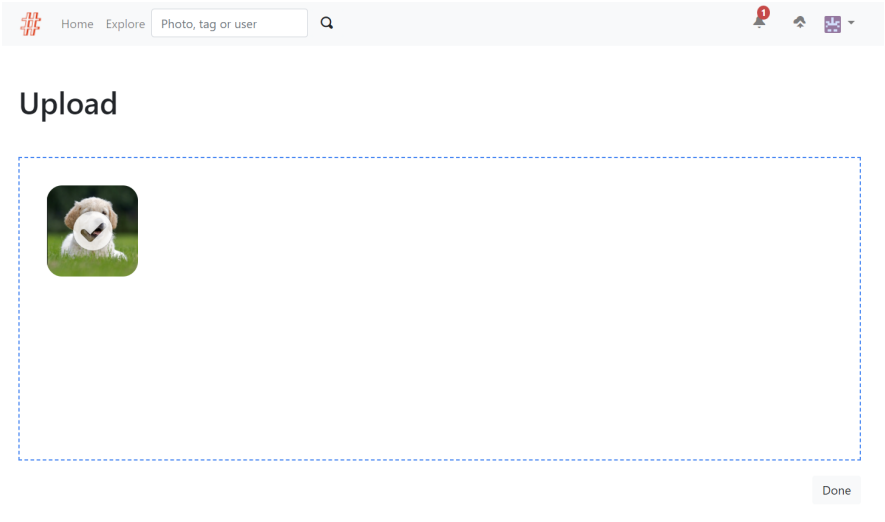
  This resultant alt text is directly passed in the creation of the Photo class object at line 137.

Interface design:

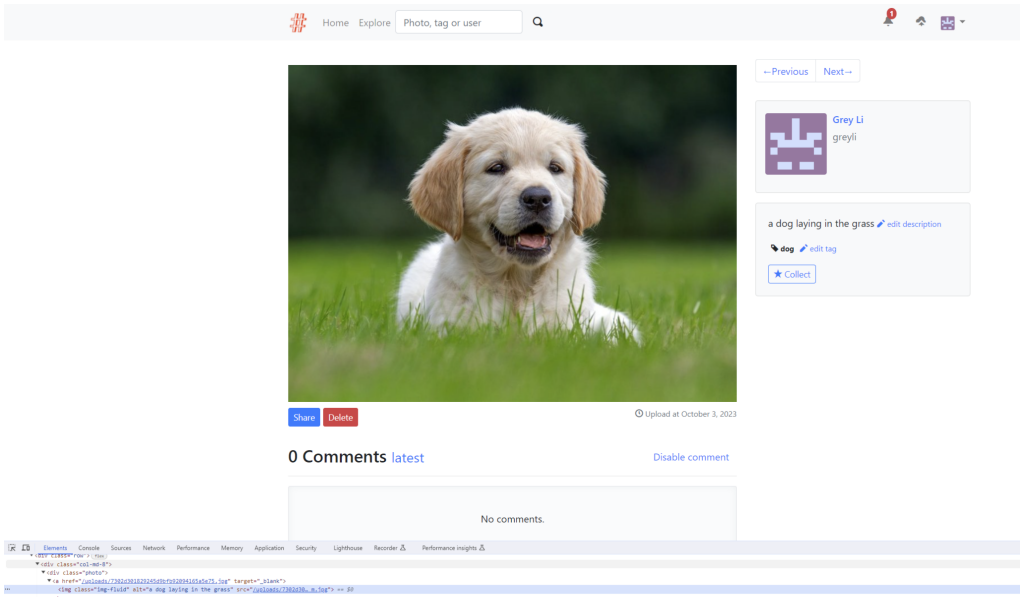
1. Click the “upload” icon on the top right corner. It should redirect to the following page.



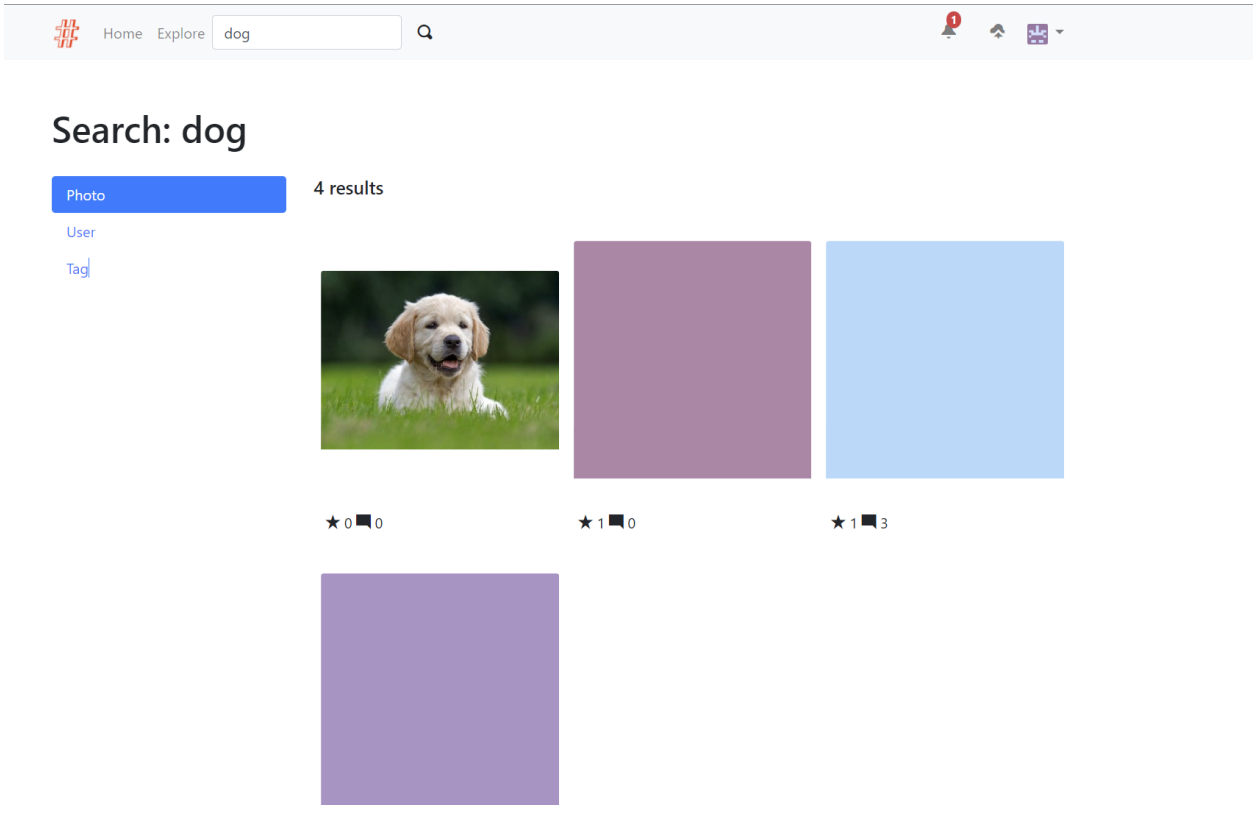
2. Click the dashed border to select images from the file system or directly drop images into the dashed border.
3. Wait until every image has a “checkmark” as shown, then click “Done”.



4. Now check the images uploaded, description, which is also alternative text, and tags are generated automatically.



5. Search the keyword “dog” in the top search bar, and the image uploaded should show up.



Potential Harms:

One possible harm we can anticipate from using machine learning to generate alternative text is cultural insensitivity. For example, prayer is widespread among multiple religions like Judaism, Islam and Buddhism. If someone uploads an image about praying, it is possible that the machine learning model identifies the wrong religion and generates an inappropriate description. In addition, there are many territorial disputes around the world, so when uploading maps, the model might output something that offends the people of a country. To mitigate this harm, assuming we treat the model as a black box, that is, we can not control the data used for training or training process, we could first provide users with a disclaimer to warn them that descriptions are generated by the machine learning model that may not have cultural sensitivity. In addition, we could have a reporting system so that users could report any generated descriptions that make them feel offended. Last but not least, based on these reports, we could build a keyword filter system to detect any word in the generated description before showing outputs to users and replace them with more generic words or even block them.

Another possible harm is that the model might have biases and stereotypes. Hendricks et al. (2018) found that some machine learning models use contextual cues on computer vision tasks, which may result in stereotypes. One example is that when the model identifies a computer, it assumes that a man is using the computer even though the image is about a woman sitting in front of a computer. Again if we treat the model as a black box, the solutions mentioned before are useful to mitigate this harm as well, but bias and stereotype can be reduced during the model training process. So we should have our audit team search for images in which the model will have a higher chance of generating biased results, and then regularly test the model with these images. If we find any inappropriate outputs, we should give the machine learning team feedback and ensure they follow ethics guidelines.

Reference:
Hendricks, L. A., Burns, K., Saenko, K., Darrell, T., & Rohrbach, A. (2018). Women also snowboard: Overcoming bias in captioning models. Computer Vision – ECCV 2018, 793–811. https://doi.org/10.1007/978-3-030-01219-9_47

Production challenges:

The main challenges of putting this solution in production and scaling it to millions of users can be two folds -

1. The size of the ML model - Good, pre-trained models are often large and heavy. The BLIP model used in this project itself goes beyond 2GB. With such a huge model, it cannot be shipped along with the application which is likely to be used on mobile phones.
   Putting the model on the cloud could be a possible solution, but will then often require a lot of time to fetch the result while uploading the image and large internet bandwidth.

2. The latency of inference: The model used produces quite accurate results but it takes a significant amount of time to generate it. This disrupts the user experience of the overall app and will be a big problem on scale.
   In order to tackle this problem, we could use a much lighter model that produces good enough results but with much less inference time so that it doesn't make the user feel a difference in the normal usage of the app without this ML feature.
   This can be a tradeoff between producing correct/accurate results versus how quickly can we generate them.

   Having a smaller model could also solve the ML model size problem which then could be shipped along with the app itself if it's compact enough.

3. If we plan to ship the model with the app itself, we could face a problem of 'how to update the model' with new, varied data. With time, there occurs a domain shift in the kind of data we are treating our model with and this could deteriorate its performance. Thus, we need to ensure we have an accessible way to update our models with the latest training.

On a side note, handling the *harms* at million level scale is a challenge in itself.

Thus, the production challenges are multifold and intertwined. We need to discuss, analyse and thoroughly understand each of them, and weigh their pros and cons before jumping into putting some ML features into an existing system at scale.