UNIVERSITÀ DI PISA

Master's degree in Artificial Intelligence and Data Engineering

Business and Project Management project documentation

ADVERTISEMENT GENERATOR

Domenico D'Orsi, Denny Meini

A.Y. 2022/2023

GitHub repository: https://github.com/dennymeo99/Advertisement_generator

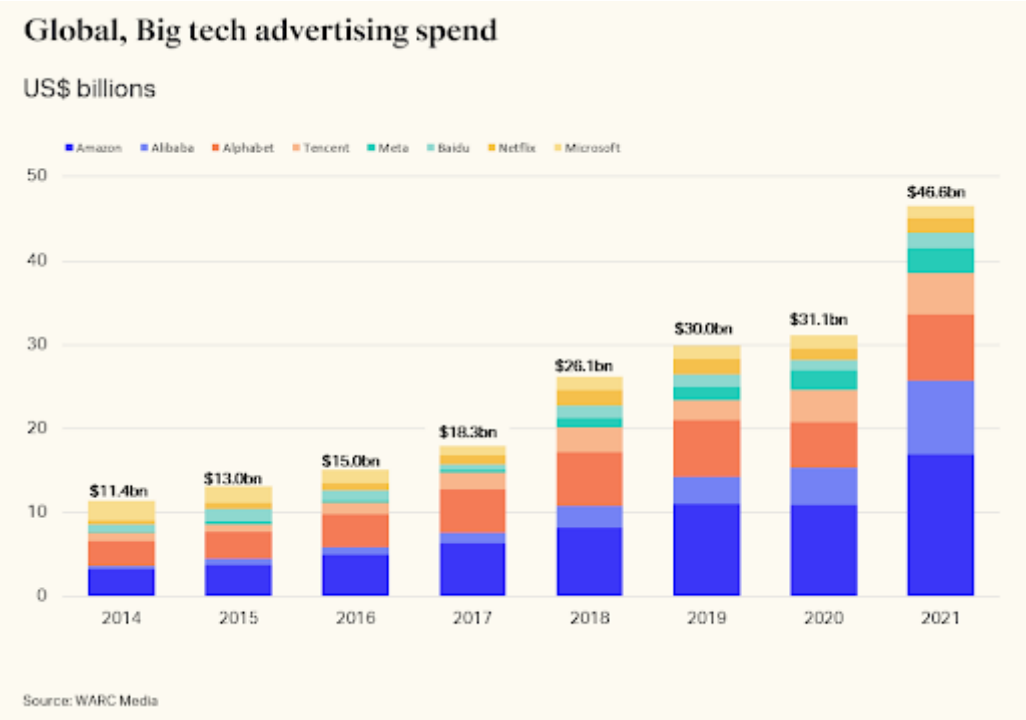
Index

1. Introduction

In the history of marketing a crucial role has always been played by advertisement. The ability of a company to advertise well their products it's an important and powerful marketing tool, since customers are strongly attracted by a captivating and impressive advertisement, which is capable to underline the unique quality of the product, that the customer can have only buying it instead of concurrency's one.

Another important benefit of a good advertisement campaign is to increase the popularity and the image of your brand, in order to make the people think immediately to your company as soon as they want to buy that type of product.

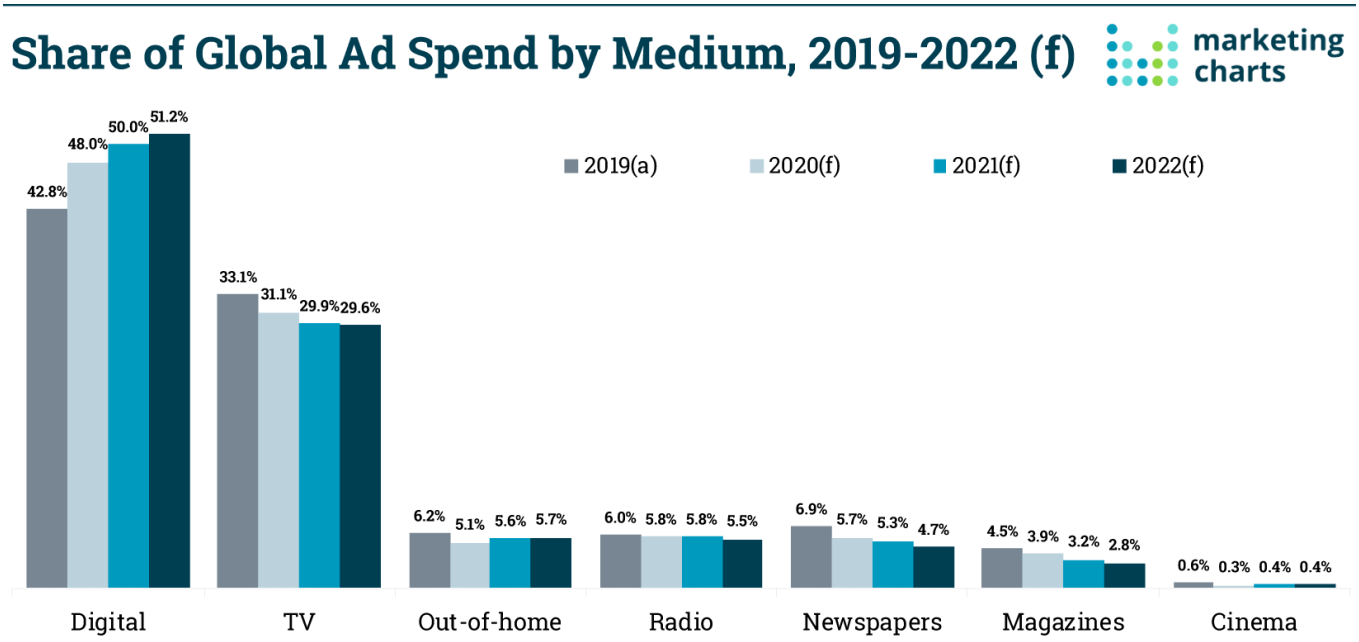
A demonstration of this is the huge amount of money that the big companies spend on their advertisement campaigns. In 2022 WARC made a study about the cost that companies had for advertisement: it results that in 2021 8 between the biggest tech companies (Amazon, Alibaba, Alphabet, Tencent, Meta, Baidu, Netflix and Microsoft) spent together the impressive sum of 46.6 billion of dollars in advertisement, with only Amazon having an expense of 16.9 billion of dollars.



Is interesting to see that in only one year the total cost increased by 15.5 billion of dollars, having an increment of 49.4%, which tells us that nowadays is fundamental to have a well-built advertisement campaing for a company, since it's the main way to increase the sales, strengthen the brand personality and companies are willing to spend a lot for it.

Talking about the advertising expenditure by media, it's very interesting to see that a large percentage of the expenditure regards internet and digital channels, as we can see from the following diagram (taken from "*marketing charts*" website)

In fact, digital channels covers slightly more than 50% of the expenditures after an 8% growth over the last few years, so it's a very large piece of the cake.



Published on MarketingCharts.com in February 2021 | Data Source: dentsu

*Dentsu's forecasts are based on its local market expertise and data collated from its brands | *"The forecasts are based on the assumption that an effective global vaccination programme will be rolled out in 2021, enabling key media events to take place."*

Those numbers really give us an interesting insight about the way companies invest in advertisements, and how important the online part of those campaigns has become.

For internet searching is very important to exploit SEO keywords. SEO stands for "Search Engine Optimization" and indicates a group of strategies that help a website to be found on internet researches, so the SEO keywords are that keywords that indicates a website and make it more reachable from users, making a specific web page appear in the first results for some given researches, even if these words are used inside the metadata or inside the text of a webpage.

We tought that inserting those words in the slogan would bring more researches towards an hypotetical website on which the slogan would then be put.

All these elements made us thought that it would be very interesting to have an application which is able to generate slogan (also using SEO keywords) and banner for a company knowing the name and the type of product it want to sell, so we implemented Advertisement generator in a view of strong

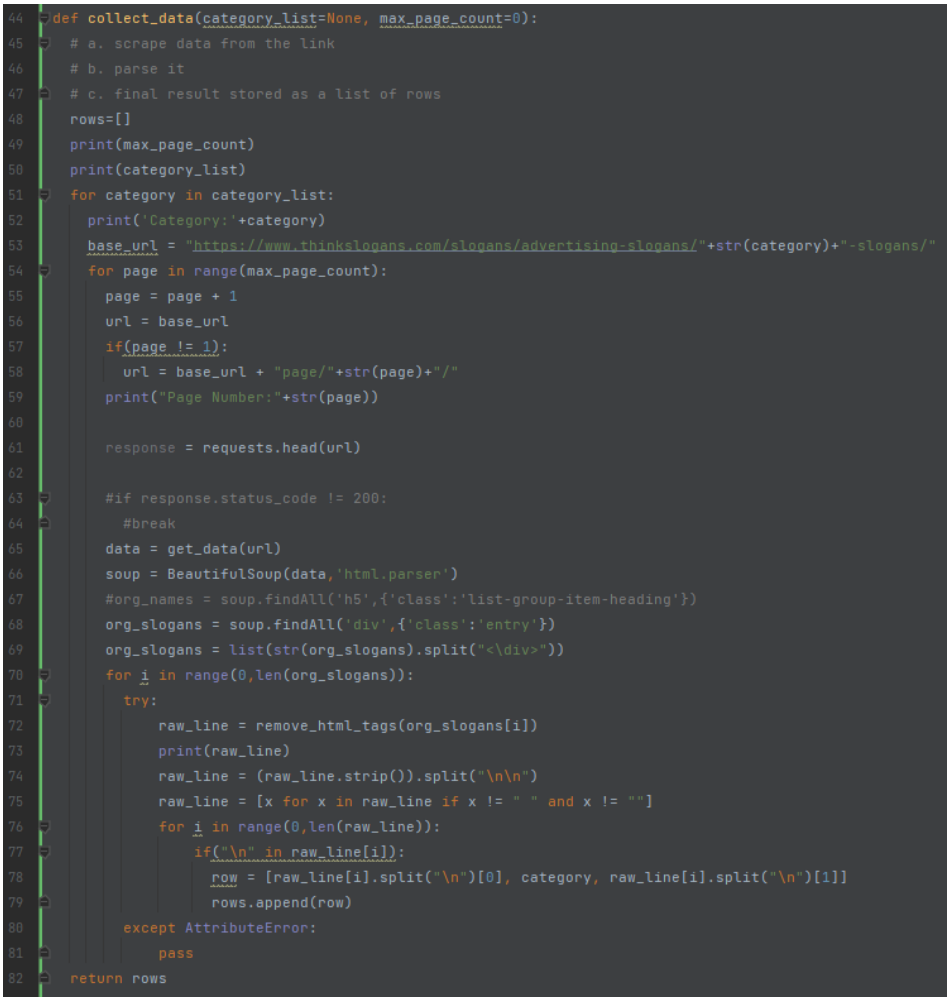
reduction of costs for this kind of operations inside a company, also using the powerful tool that generative AI represents.

2. Description of the process

Our work can be divided into 2 macro-tasks: the generation of the slogan and the creation of the banner. In both cases we first had to perform a scraping process in order to extract from web some words and images, then we used them to create two datasets (one for the slogans and one for the images). Afterwards we had to train the 2 models using the dataset we just created. Finally we could test the models and use them in the final applications, which create the slogan and the banner starting from the name of the company and the category of the product. Furthermore we asked to a popular artificial intelligence, OpenAI, to generate itself a slogan and a banner starting from the same input we gave to our models, so that we could compare the results.

2.1 Slogan model training

Before to start we have to install the requiremens with the command pip install -r requirements.txt (the file requirements.txt is part of the repository). Our slogan generation work started with the online scraping of the slogans. The code fit to do this is written in the Python notebook scraping.ipynb,



```
def collect_data(category_list=None, max_page_count=0):
  # a. scrape data from the link
  # b. parse it
  # c. final result stored as a list of rows
  rows=[]
  print(max_page_count)
  print(category_list)
  for category in category_list:
    print('Category:'+category)
    base_url = "https://www.thinkslogans.com/slogans/advertising-slogans/"+str(category)+"-slogans/"
    for page in range(max_page_count):
      page = page + 1
      url = base_url
      if(page != 1):
        url = base_url + "page/"+str(page)+"/"
      print("Page Number:"+str(page))

      response = requests.head(url)

      #if response.status_code != 200:
        #break
      data = get_data(url)
      soup = BeautifulSoup(data,'html.parser')
      #org_names = soup.findAll('h5',{'class':'list-group-item-heading'})
      org_slogans = soup.findAll('div',{'class':'entry'})
      org_slogans = list(str(org_slogans).split("<\div>"))
      for i in range(0,len(org_slogans)):
        try:
            raw_line = remove_html_tags(org_slogans[i])
            print(raw_line)
            raw_line = (raw_line.strip()).split("\n\n")
            raw_line = [x for x in raw_line if x != " " and x != ""]
            for i in range(0,len(raw_line)):
                if("\n" in raw_line[i]):
                  row = [raw_line[i].split("\n")[0], category, raw_line[i].split("\n")[1]]
                  rows.append(row)
        except AttributeError:
            pass
  return rows
```

which contains several modules. The first module allowed us to generate the textual database that we used in order to train the model. In this module we used the Python library BeautifulSoup that contains some methods for the analysis of HTML and XML documents, so it was perfect for our aim. We took the words from the website “Think Slogans”, which contains a lot of slogan regarding different genres of products and companies.

The key function of the module is “collect_data”, in this function we scrolled the HTML document of the website’s list of slogans for some predefined categories (for example airlines, beverage, fast food), we collected data and then, using BeautifulSoup, we parsed them and finally we obtained a list of rows containing the slogan, the category and the company. The last operations of the module was to write the rows in a dataset and preprocess it in order to make it ready for the training phase.

The second Python notebook we wrote is textual_model_training.ipynb. Using this notebook we trained the model we chose for the automatic generation of the slogan, which is “Google flan t5 small”, we chose this model because it was quite simple and good for small-sized datasets, but it still gave us good results. The first operations were to encode the training set and tokenize it in a specific format, with the addition of a padding, these tasks were done in the function “encode_batch” in the way shown in the image below.

```
def encode_batch(examples):
    # the name of the input column
    text_column = 'question'
    # the name of the target column
    summary_column = 'answer'
    # used to format the tokens
    padding = "max_length"

    inputs, targets = [], []
    for i in range(len(examples[text_column])):
        if examples[text_column][i] and examples[summary_column][i]:
            inputs.append(examples[text_column][i])
            targets.append(examples[summary_column][i])

    inputs = [prefix + inp for inp in inputs]
    model_inputs = tokenizer(inputs, max_length=512, padding=padding, truncation=True)
    labels = tokenizer(targets, max_length=512, padding=padding, truncation=True)

    # rename to labels for training
    model_inputs["labels"] = labels["input_ids"]

    return model_inputs
```

The tokenizement job was completed into the “load_split” function, which also set the dataset format to torch.

```python
def load_split(split_name, max_items):
    # load the split
    dataset = load_dataset('csv', data_files='/content/preprocessed.csv', delimiter=',')[split_name]
    # only use the first max_items items
    dataset = dataset.filter(lambda _, idx: idx < max_items, with_indices=True)
    # tokenize the dataset
    dataset = dataset.map(
        encode_batch,
        batched=True,
        remove_columns=dataset.column_names,
        desc="Running tokenizer on " + split_name + " dataset",
    )
    # set the format to torch
    dataset.set_format(type="torch", columns=["input_ids", "labels"])

    return dataset
```

Finally we could train the model properly using “transformers”, a Hugging Face library which provided us some API that made us able to train the model. In particular we used TrainingArguments and Trainer, in order to set the arguments for the training and do it. We also exploited Low Rank Adaptation (LoRA), which semplifies this kind of trains.

```python
# start with the pretrained base model
model = AutoModelForSeq2SeqLM.from_pretrained(
    base_model
)

# set the parameters for LoRA
config = LoRAConfig(
    r=8,
    alpha=16,
    intermediate_lora=True,
    output_lora=True
)


# small batch size to fit in memory
batch_size = 1

training_args = TrainingArguments(
    learning_rate=3e-4,
    num_train_epochs=1,
    per_device_train_batch_size=batch_size,
    per_device_eval_batch_size=batch_size,
    logging_steps=200,
    output_dir="./training_output",
    overwrite_output_dir=True,
    remove_unused_columns=False
)

trainer = Trainer(
    model=model,
    args=training_args,
    tokenizer=tokenizer,
    train_dataset=load_split("train", 1000),
)

trainer.train()
```

2.2 Images model training

For which regards the images dataset we needed to repeat the same steps we did for the slogans dataset, starting from the scraping of the images which we did with a function in the scraping.ipynb notebook. The scraping was performed on https://people.cs.pitt.edu/~mzhang/image_ads using the library "request" and permitted us to obtain the images dataset.

```
import requests

for i in range(0, 5):
  j = 0
  for j in range (0, 100000):
    base_url = f'https://people.cs.pitt.edu/~mzhang/image_ads/{i}/{j}.jpg'
    if(requests.get(base_url).status_code != 404):
      print("Immagine ottenuta: " + base_url)
      img_data = requests.get(base_url).content
      filepath = "/content/advertisements/ad_" + str(i) + "_" + str(j) + ".jpg"
      with open(filepath, 'wb') as handler:
          handler.write(img_data)
    else:
      print("immagine non ritrovata: " + base_url)
```

Then we can start with the train the model, which is "Stable Diffusion v1.5" from Hugging Face.

The first steps in this sense were the setup to the actual training: we had to perform the of import some model-weights converting tools and logging to HuggingFace in order to use a pretrained model, then we had to specify the training set folders path as well as various parameters for the training itself, *(learning rate, batch size, train steps ecc.).*

The training phase, finally, is up to a script made available by HuggingFace itself, called "*train_dreambooth.py*" (https://raw.githubusercontent.com/ShivamShrirao/diffusers/main/examples/dreambooth/train_dreambooth.py), which exploits the DreamBooth class itself.

Our configuration is shown below:

```
!python3 train_dreambooth.py \
  --pretrained_model_name_or_path=$MODEL_NAME \
  --pretrained_vae_name_or_path="stabilityai/sd-vae-ft-mse" \
  --output_dir=$OUTPUT_DIR \
  --revision="fp16" \
  --with_prior_preservation --prior_loss_weight=1.0 \
  --seed=1337 \
  --resolution=512 \
  --train_batch_size=1 \
  --mixed_precision="fp16" \
  --use_8bit_adam \
  --gradient_accumulation_steps=1 \
  --learning_rate=1e-6 \
  --lr_scheduler="constant" \
  --lr_warmup_steps=0 \
  --num_class_images=50 \
  --sample_batch_size=4 \
  --max_train_steps=800 \
  --save_interval=10000 \
  --save_sample_prompt="photo_of_advertisement" \
  --concepts_list="concepts_list.json"
  --train_text_encoder \
```

Then, some preview images are generated before moving on to the model test part

```
import os
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

weights_folder = OUTPUT_DIR
folders = sorted([f for f in os.listdir(weights_folder) if f != "0"], key=lambda x: int(x))

row = len(folders)
col = len(os.listdir(os.path.join(weights_folder, folders[0], "samples")))
scale = 4
fig, axes = plt.subplots(row, col, figsize=(col*scale, row*scale), gridspec_kw={'hspace': 0, 'wspace': 0})

for i, folder in enumerate(folders):
    folder_path = os.path.join(weights_folder, folder)
    image_folder = os.path.join(folder_path, "samples")
    images = [f for f in os.listdir(image_folder)]
    for j, image in enumerate(images):
        if row == 1:
            currAxes = axes[j]
        else:
            currAxes = axes[i, j]
        if i == 0:
            currAxes.set_title(f"Image {j}")
        if j == 0:
            currAxes.text(-0.1, 0.5, folder, rotation=0, va='center', ha='center', transform=currAxes.transAxes)
        image_path = os.path.join(image_folder, image)
        img = mpimg.imread(image_path)
        currAxes.imshow(img, cmap='gray')
        currAxes.axis('off')

plt.tight_layout()
plt.savefig('grid.png', dpi=72)
```

And at last, the inference part used to produce the banners given the advertisement prompt

```
prompt = "advertisement banner for an airline company called Apulian Airlines"
negative_prompt = ""
num_samples = 4
guidance_scale = 7.5
num_inference_steps = 24
height = 512
width = 512

with autocast("cuda"), torch.inference_mode():
    images = pipe(
        prompt,
        height=height,
        width=width,
        negative_prompt=negative_prompt,
        num_images_per_prompt=num_samples,
        num_inference_steps=num_inference_steps,
        guidance_scale=guidance_scale,
        generator=g_cuda
    ).images

for img in images:
    display(img)
```

The result will be 4 different images realized by our custom trained model. For sake of simplicity and to strongly decrease the final repository size, the model was saved on a HuggingFace repository, and it will be exploited by an API call

2.3 Slogan and banner generation

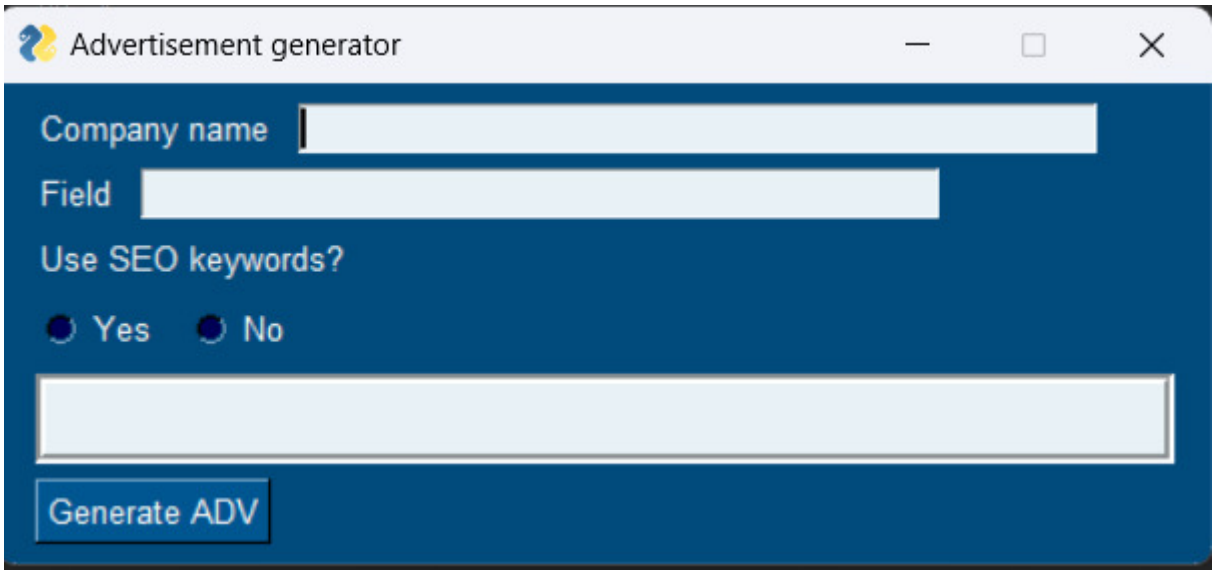
With both the models trained we could write an application that used them to generate a slogan and a banner. The application is written in python and is composed of 4 files: “main_page.py”, “result_page.py”, “inference.py” and “SEO_keyword.py”.

The first of these modules, “main_page.py”, generates the GUI of the application, which is a single window for the insertion of the inputs; this module is responsible to call the other three too.

The module “result_page.py” shows a window which contains the results of the generation process (this will be discussed more deeply in section 3).

The module “inference.py” is responsible for the effective creation of the slogans and the images: it performs a sequence of API call to OpenAI, exploiting chatGPT, DALL-E and our pretrained models.

The module “SEO_keywords” uses a tool which searches on internet some keywords used in online researches correlated to the input and returns them on the result page. The module uses this tool in order to research the keywords: https://github.com/chukhraiartur/seo-keyword-research-tool..

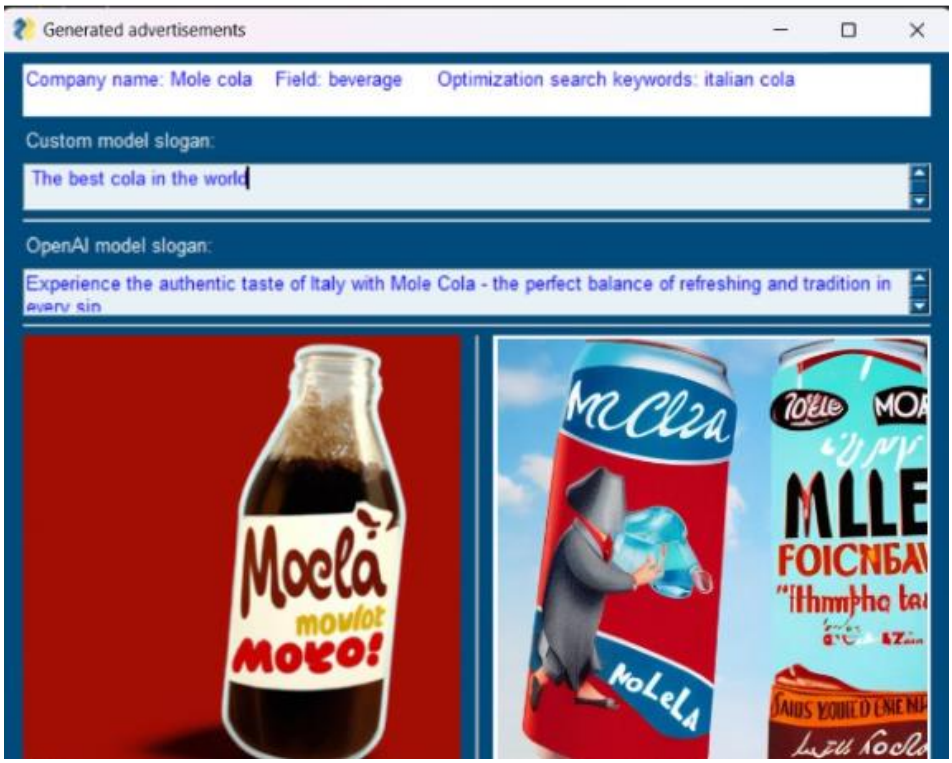


The one above is how the homepage looks, the user has to insert the company name, the field of the company and choose if use SEO keywords, then by clicking on Generate ADV the application starts.

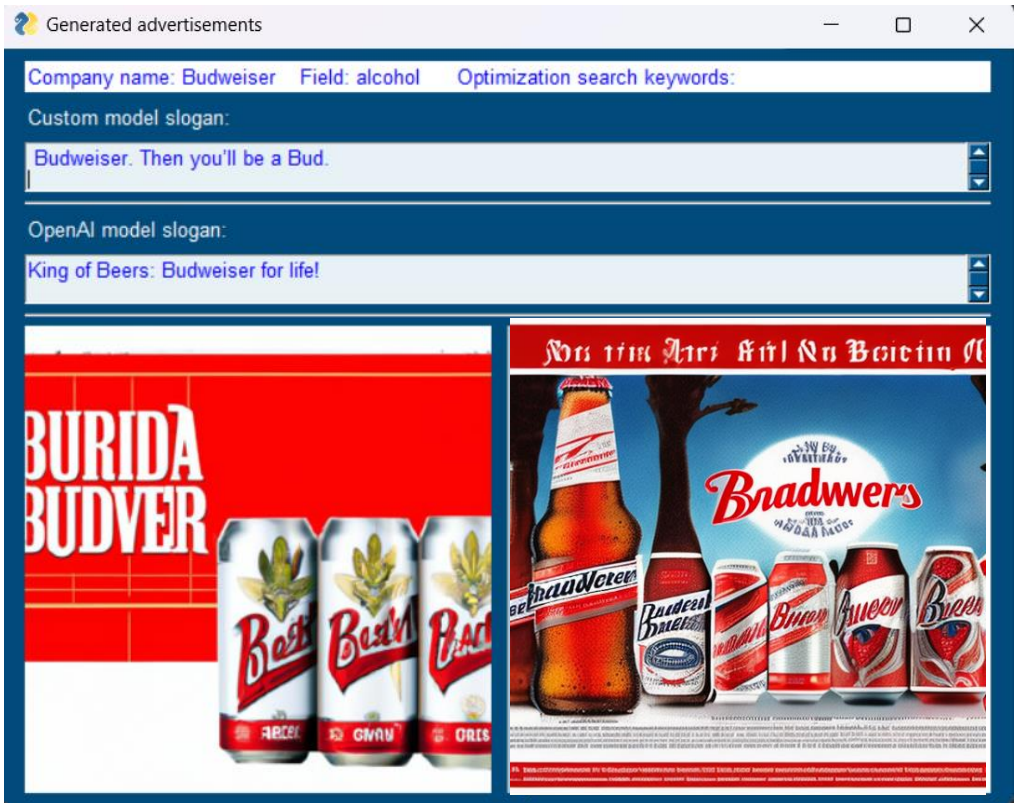
3. Description of the results

We proceed here to analyze the results comparing the slogans and the images produced by the models with the one that OpenAI provided us. In the following images we can see the 2 slogans and the 2 banners (the OpenAI banner is the one on the left, the custom model ones are on the right).

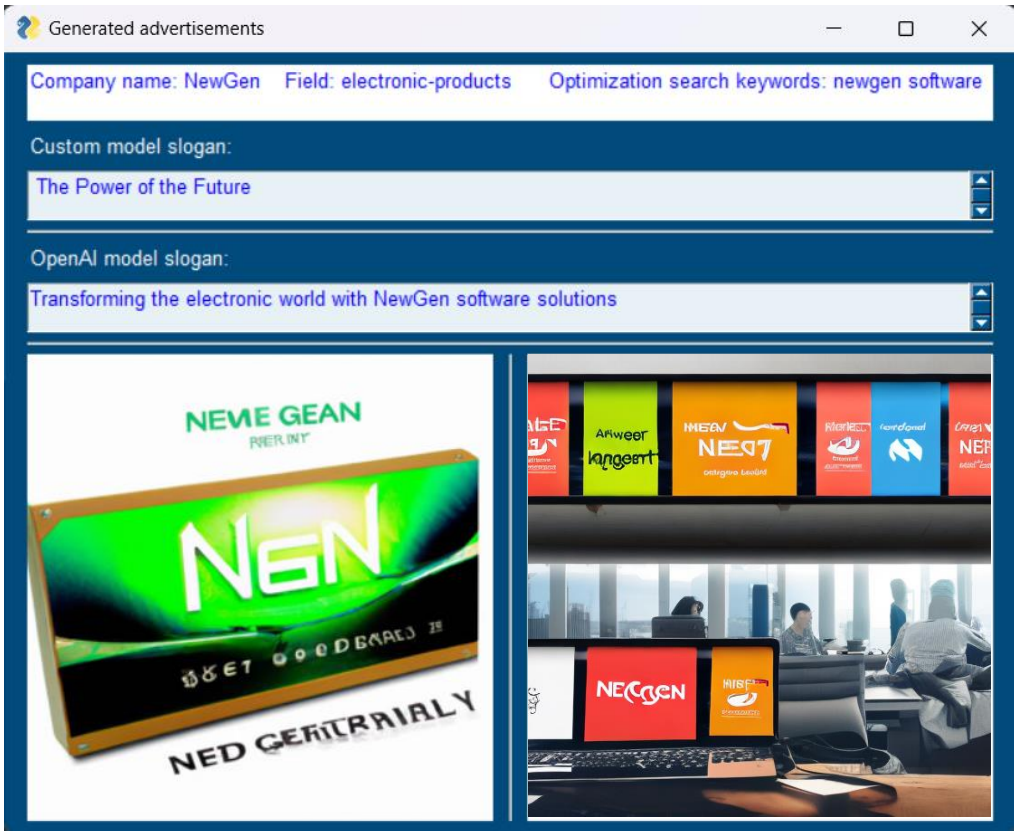
- Mole cola, beverage (existing brand)



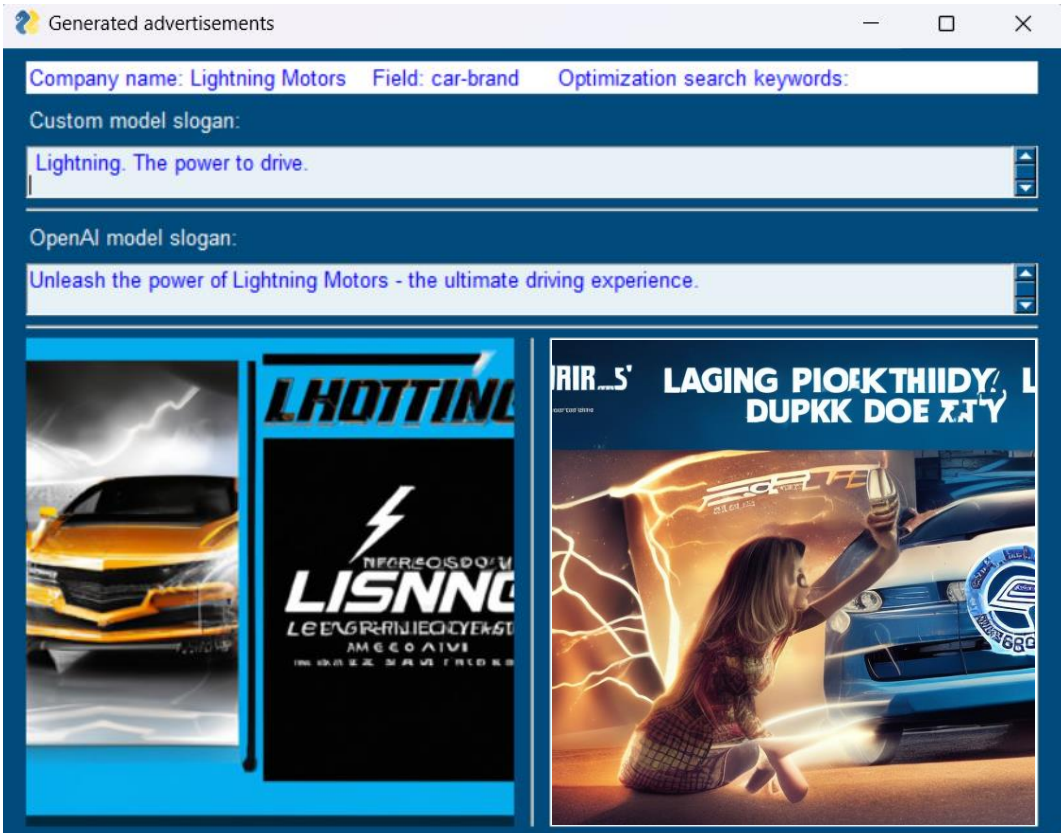
- Budweiser, alcohol (existing brand)



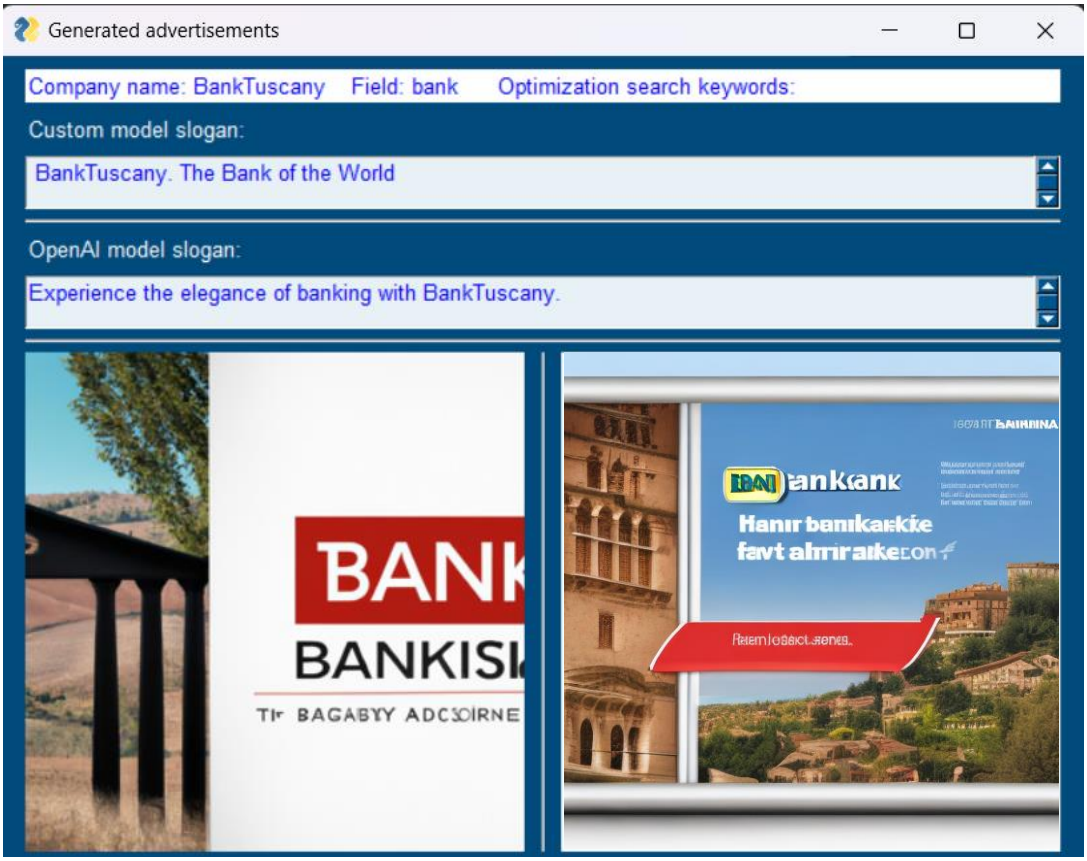
- NewGen, electronics (fictional brand)



- Lightning Motors, car-brand (fictional brand)



- BankTuscany, bank (fictional brand)



For what regards the slogan we can observe that in some cases the slogans generated by our model are simple but good, they reach the goal to talk good about the product and be catch yat the same time. Here the main problem is about the model producing slogans a little bit too general, but this is clearly due to the small size of our dataset for the given task and model.

Talking about the banners we see that globally the application has a good performance, is absolutely natural that the quality of the images generated is not brilliant, because this model needs a very big dataset in order to give optimal results; moreover the words are not correctly defined because of the limitations of these tools. Anyway, our application returns images that are coherents with the inputs and represent correctly the desired product, so it's still a very useful application, capable to create a slogan and a banner that advertizes actually the product, or at least are a really good starting point for an expert team.

In both cases, OpenAI offers two models with really good performances for an affordable price, in particular if the customer is a company.

4. Conclusions

We think that the application could receive some developements and improvements in order to give more accurate results, for example an interesting feature could be to allow the user to specify the country of the company or the age range of the possible customers. In that way both the slogan and the banner would be richer of details and would have a powerful impact on the possible customers, producing a more targetized kind of advertisement.

Finally we can say that this application is able to provide good results, even if not exceptional, in fact, all the slogans and the banners we obtained (both with the pretrained models and with OpenAI) are correct. Of course the application can be improved collecting more data, for example from the companies we are partner with, but it's still a useful tool, capable of represent the product in a satisfying way, and giving interesting starting point to an actual ADV team inside the company, improving and fastening this process thus reducing costs. Moreover, it's clear that the generative AI can be a very powerful tool, capable of helping our company to be recognized world wide thanks to a very efficient advertising campaign.