Machine Learning and Content Analytics

Real Estate Ad Quality Assistant

Instructor: Haris Papageorgiou

Team

First Name	Last Name	Student Number	Email
Konstantinos	Toulis	f2822313	kon.toulis@aueb.gr
Ioannis	Tsougkrianis	f2822314	ioa.tsougkrianis@aueb.gr
Athanasios	Kostarelis	f2822305	ath.kostarelis@aueb.gr
Viktor	Kalatzis	f2822318	vik.kalatzis@aueb.gr

TABLE OF CONTENTS

1. Introduction

1.1 Project Overview

Real estate and property listing websites offer an overabundance of content in terms of images and text used to describe a house to a potential buyer, yet few of them leverage the powerful tools of modern deep learning tools in order to refine said content. In this project, we focused on offering an AI tool that would enhance the user experience for both property owners creating a listing and potential buyers for one of the biggest real estate aggregators in Greece, Spitogatos.

1.2 Our Project

House Description and Feature Comparison

Here the main focus is to create a description for the whole house. We achieve that by first taking each picture from the ad and generating a description for each. That description was prompted to focus on some characteristics such as room type, lighting, floors and furniture. After that we combine all the descriptions into one that describes the whole house. The new description includes the number and the type of the rooms, some noteworthy features, the flooring type and lastly the lighting. Additionally, by using prompt engineering, we create a new enhanced and improved version of the description, by specifying in the prompt what information we want the new description to include. Lastly, we take the enhanced total summary, compare it to the description given by the house owner, calculate a similarity between the two and finally we generate an improved description that fits the property better.

House Description Feature Extraction

With this implementation, our objective is to recognize whether certain important features exist in a property's description in order to assess tis quality. We concluded that for a listing to have a complete description, four key aspects should be mentioned. Those were public transport options, various amenities (such as distance to grocery store, park etc.), the layout of the house (rooms and how they are connected) as well as the house type (apartment, whole building etc.). For this purpose we fine-tuned an LLM which successfully identifies and displays these features.

Price Prediction

Our goal here is to predict the price of a real estate entity given descriptive, categorical, and image data that we had extracted from the house listing in our dataset. We assessed that since price correlates to a house's quality, this should be reflected on the uploaded content

of the user, mainly the given description and images. Model development therefore focused on applying various machine learning techniques on the text data from the property descriptions and the uploaded images.

1.3 Our Vision/Goals

Our team's vision is to increase the overall quality and usability of real estate listing, by deploying advanced machine learning techniques. We aim to enhance the user experience for both the buyer and the seller, by enhancing the descriptions of the house ads in order to better reflect their features. We want the buyer to be able to easily understand key aspects of the property he or she is interested in, and the seller to be able to present their property more efficiently and clearly, which will result in higher interest for the ad. We also attempt to create a price classification on listings based on the uploaded images and description. Our hypothesis is that such a classification could provide insight as to whether the user uploaded content matches the house quality. By applying this model we could perhaps better moderate the content of the platform, inform sellers if their listing should be improved or potentially assist buyers in finding undervalued houses.

2. Data Collection, Preprocessing, and Model Development

Overview of Dual Model Approach

Early on, it became clear that the description enhancement and price prediction tasks should be handled by separate models. The first task can be more effectively tackled with an LLM that is specifically engineered for it. The second can be addressed with a more compact model we can develop from scratch. Each of the two models, focused on different aspects of the real estate process: one focuses on enhancing and verifying property descriptions, while the other is focusing on the pricing of the listing.
For this reason we split our focus into two fronts. The first would be to fine-tune an LLM for the purposes of enhancing listing descriptions with attributes extracted from its images. We also wanted to extract important information from the listing's description, in order to assess its quality. These desired features will be explained in the following paragraphs. The second approach would be to implement a price prediction model, through deep learning techniques showcased through the course of the semester.

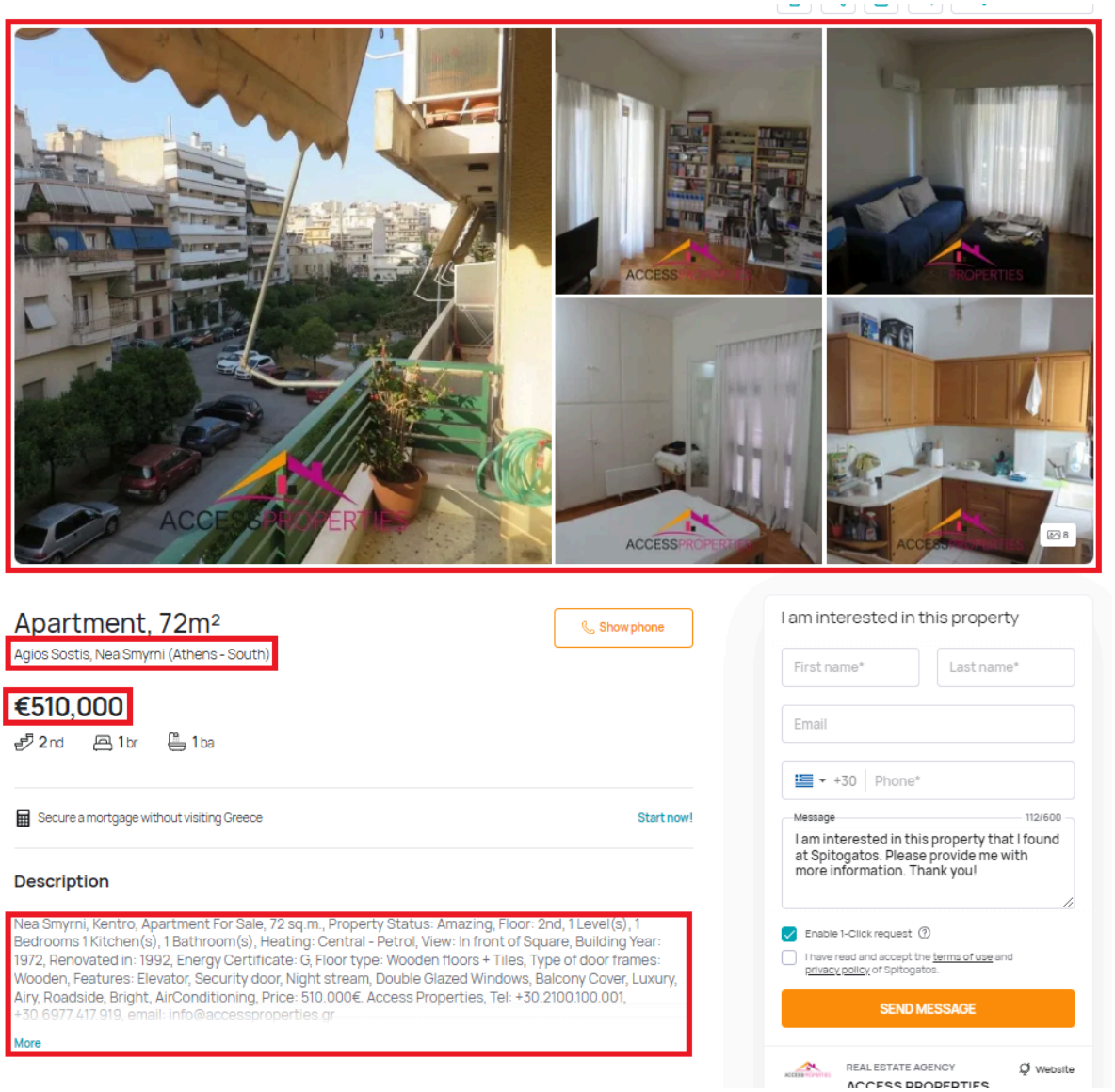
2.1 Data Collection

2.1.1. Data Sources

All data for this project was collected using a scraper from the Greek real estate website https://www.spitogatos.gr/. We used that site because it included a variety of listings from houses from different areas of Athens, Greece, providing a diverse dataset for analysis. Each of the four members of our team collected house listings from near their residence area, in order to have a diverse, complete but also a realistic dataset. We did not want to include only the average house ad in our dataset. Our aim was to have listings of good (luxury houses), average (everyday houses) and bad houses. Additionally to the quality of the house, we focused on the quality of the ad itself. The metric to rate the listing, was based on a couple criteria. First of all, the number of the images and their quality and then the quality of the description based on the included information that we could get, but also based on how well written it was.
Also the ads are written in English, which helps the ML models to have more accurate results, but also gives us the freedom to pick and choose the appropriate model for our tasks, from a much wider selection of ML models that were pre-trained on English datasets, compared to utilizing models that could work ok a dataset in Greek.
For our project, we utilized 600 ads of properties as the base for our dataset. As it is written later on in the report, we took advantage of some data augmentation techniques, to achieve better results, going from 600 to about 2500 ads total.

2.1.2 Web scraper

For the creation of a dataset that our project can be built upon, we developed a web scraper. We wanted our dataset to consist of ads of houses from various neighborhoods of Athens, of different prices and conditions in order for our models to be able to gain better information and make better predictions.
After our team members had finished manually collecting links from ads in an XLSX file, a custom web scraper was developed, mainly utilizing the selenium python library. The main functionality of the scraper, without getting into many technical details, was to navigate the provided links from the XLSX file, and record some details for each ad and save them in a CSV file. The details that we recorded were the description of the property, location, price and the URLs of the associated images from the ad.



The next step was to create a different Python script, that would take as input the URLs of the images from the previous CSV file and would download and store in a local drive, to make sure that we always have direct access to the images for our analysis. The difficulties that we faced here mainly had to do with getting flagged by CAPTCHA. Real estate sites are

notorious for being targeted by web scrapers, as many attempt to profit from the information posted there. For that reason, they usually have a pretty robust bot flagger to minimize non-human activity and traffic. In order to bypass that problem there were a few techniques that we used. We mainly simulated human behavior by adding a small delay between the frequency of each request. Additionally to the delay, we added some random clicking, browsing around and scrolling into the scraper, which resulted in greatly decreasing the chances of CAPTCHA blocking us.

2.1.2.1. Scraper Implementation (scrapper.py)

The web scraper was developed by using Python in combination with the Selenium library. That library helped a lot in the browser interactions. It gave us the option to navigate through the different house listings, to interact and extract the specific elements and data that we thought to be useful in our dataset.
Another useful library was the fake_useragent library. By utilizing it, we were able to copy quite a few random human-like behaviors and avoid getting flagged by CAPTCHA, that otherwise would apply to us scraping limitations and effectively would not allow us to create the main dataset for our project.

The scraper is initialized by the function “get_driver” where its main purpose is to set up and configure some browser automations and different options (one of them being “user agent” from the library “fake_useragent”). Additionally it initializes and returns a Chrome WebDriver.

```
headers_list = [
    {
        'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.1
    {
        'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.1
    {
        'User-Agent': "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/92.0.4515.1
    {
        'User-Agent': "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/126.0.0.0 S
]

ua = UserAgent()


1 usage
def get_driver():
    chrome_options = Options()
    chrome_options.add_argument(f"user-agent={ua.random}")
    driver = webdriver.Chrome(options=chrome_options)
    return driver
```

Next, the function “simulate_human_interaction” was developed in order to recreate behaviors of a human user, to avoid detection as a bot. Its main tasks are to do random movements and not avoid doing movements in perfect straight line, as bots do, using the ActionChains. It also implemented some random scrolling on the page, at random moments.

```python
def simulate_human_interaction(driver):
    actions = ActionChains(driver)
    for _ in range(random.randint(1, 3)):
        actions.move_by_offset(random.randint(0, 10), random.randint(0, 10)).perform()
        time.sleep(random.uniform(0.5, 2))
    driver.execute_script("window.scrollBy(0, {})".format(random.randint(100, 500)))
```

The main function of the scraper is the “scrape_real_estate(urls)” where it is scraping the data from the different ads that we provide through a list of URLs. Then it loads the site cookies whenever that is possible, it extracts the key elements that we have specified we want, such as the URLs of the images from the listing, the location, description and price. In between extraction of these features, it waits 20 seconds before moving on to the next listing, in order to avoid detection by CAPTCHA.

```python
def scrape_real_estate(urls):
    driver = get_driver()

    for url in urls:
        try:
            driver.get(url)
            cookies_file = 'cookies.json'

            if os.path.exists(cookies_file):
                try:
                    load_cookies(driver, cookies_file)
                    driver.get(url)
                except:
                    print("Could not load cookies")

            time.sleep(random.uniform(2, 5))
            wait = WebDriverWait(driver, 20)

            simulate_human_interaction(driver)

            images = wait.until(EC.presence_of_all_elements_located((By.CLASS_NAME, 'property__gallery__item')))

            location = wait.until(EC.presence_of_element_located((By.CLASS_NAME, 'property__address'))).text.strip()
            try:
                description = wait.until(
                    EC.presence_of_element_located((By.CLASS_NAME, 'property__description'))).text.strip()
            except:
                description = ""
            price = wait.until(EC.presence_of_element_located((By.CLASS_NAME, 'property__price__text'))).text.strip()
            price = fix_price(price)

            image_urls = []
```

```
            for image in images:
                try:
                    image_urls.append(image.find_element(By.TAG_NAME, 'img').get_attribute('src'))
                    time.sleep(random.uniform(0.1, 1))
                except Exception as e:
                    print(f"Error extracting data: {e}")
                    continue

            save_cookies(driver, cookies_file)
            description = description.replace('\n', ' ')
            with open("data.txt", 'a', encoding="utf-8") as file:
                file.write(url)
                for image in image_urls:
                    file.write(image + ' ')
                file.write('\n')
                file.write(description + '\n')
                file.write(location + '\n')
                file.write(str(price) + '\n')
                # file.write('\n')
            with open("success.txt", 'a', encoding="utf-8") as file:
                file.write(url)
        except:
            print(url)
```

Additionally, the two functions “save_cookies” and “load_cookies” would respectively load the cookies, of the previous session, in the beginning of the current session and save the cookies of the current session at the end of it so we can use it later, in order to not get flagged as a bot.

```
def save_cookies(driver, file_path):
    cookies = driver.get_cookies()
    with open(file_path, 'w') as file:
        json.dump(cookies, file)


1 usage
def load_cookies(driver, file_path):
    with open(file_path, 'r') as file:
        cookies = json.load(file)
    for cookie in cookies:
        driver.add_cookie(cookie)
```

Lastly, function “scrape_url(url)”, scrapes the URL of the ad, formats and saves the previous scraped data in a CSV file.

```python
def scrape_url(url):
    image_urls, price, location, description = scrape_real_estate(url)
    description = description.replace('\n', ' ')
    with open("data1.txt", 'a', encoding="utf-8") as file:
        for image in image_urls:
            file.write(image + ' ')
        file.write('\n')
        file.write(description + '\n')
        file.write(location + '\n')
        file.write(str(price) + '\n')
```

2.1.2.2. Image Processing and Data Structuring (parse_data.py)

Since the scraper was just saving the URLs of the images, we developed a separate Python script in order to download the images from their links and save them as a structured and usable format in a drive.
In the beginning, it reads a text file that has all the URLs of the images, then it creates a unique file in our directory based on the link from the ad. Moving on, we aim to handle the individual images, by downloading them and saving them in the appropriate directory for each listing and also handling cases where the download would fail.
Next, a process for structuring the data into a list begins, where every entry has the ad's link, description, location, price and the path for the directory with the listing's saved images. Then, as the last step, the structured data from the previous step, were written in a CSV file that later on we used on our project for further analysis.

2.2. Dataset Overview

2.2.1. Exploratory Data Analysis (EDA)

After gathering the data with the use of the scrapper, we conducted some exploratory data analysis (EDA) to get an overview of the composition of our variables, those being the location, the price, as well as the images.

Price:
The mean price of a property is approximately €519,589 with a standard deviation equal to €1,429,701, indicating that we have various prices in properties. The minimum price of a property is €17,000. The 25% of properties are priced below €85,000, representing the lower priced houses. The median price is equal to €180,000 and that means that 50% of the

houses are below this price. As well as, the 25% of properties are priced above €449,000, indicating higher value properties in the dataset. The highest (maximum) price is €18,000,000, showing the most expensive properties.

Descriptive Statistics for Locations (Counts of each location):

Location	
Nea Smyrni (Athens - South)	52
Kato Petralona, Petralona (Athens - Center)	40
Center, Nea Smyrni (Athens - South)	38
Ano Petralona, Petralona (Athens - Center)	38
Ano Nea Smyrni, Nea Smyrni (Athens - South)	22
…	..
Attiko Alsos, Poligono - Tourkovounia (Athens - Center)	1
Nosokomeio Pedon, Goudi (Athens - Center)	1
Goudi, Zografou (Athens - South)	1
Mouseio, Exarchia - Neapoli (Athens - Center)	1
Kastella - Passalimani (Piraeus)	1

We have 46 missing values in description.

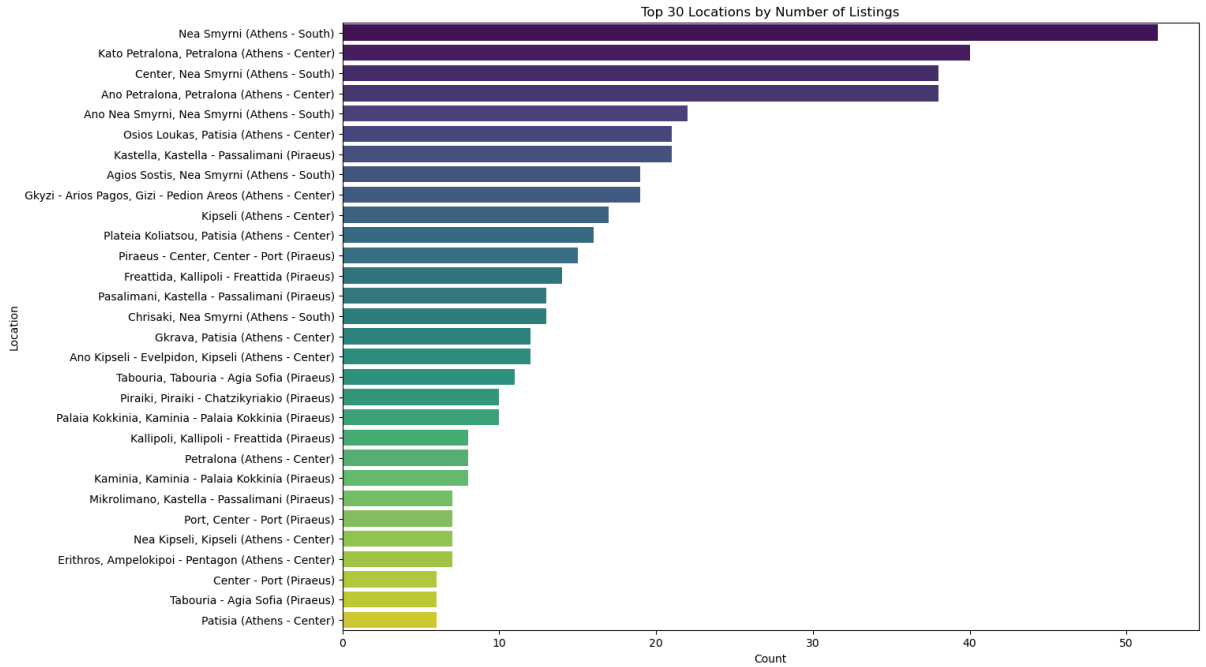


Figure 1: Top 30 Locations by Number of Listings

Firstly, we created a barplot to visualize the top 30 locations based on the number of houses in descending order, as we can see in Figure 1. In the y-axis we have the locations of the houses and in the x-axis we have the number of listings for each location. This plot helps us to understand which areas have the most listings. As we can see in the first position we have Nea Smyrni (Athens - South) with over 50 listings, more specifically 52 listings. Location Kato Petralona, Petralona (Athens - Center) follows with the number of listings equal to 40. Next, we have again Center, Nea Smyrni (Athens - South) with 38 listings. We also have many areas in Piraeus and in Patisia among the top 30 locations. So, we can realize that the

biggest number of listings are concentrated in the center or south of Athens and the fewer listings are in some other areas like Piraeus.

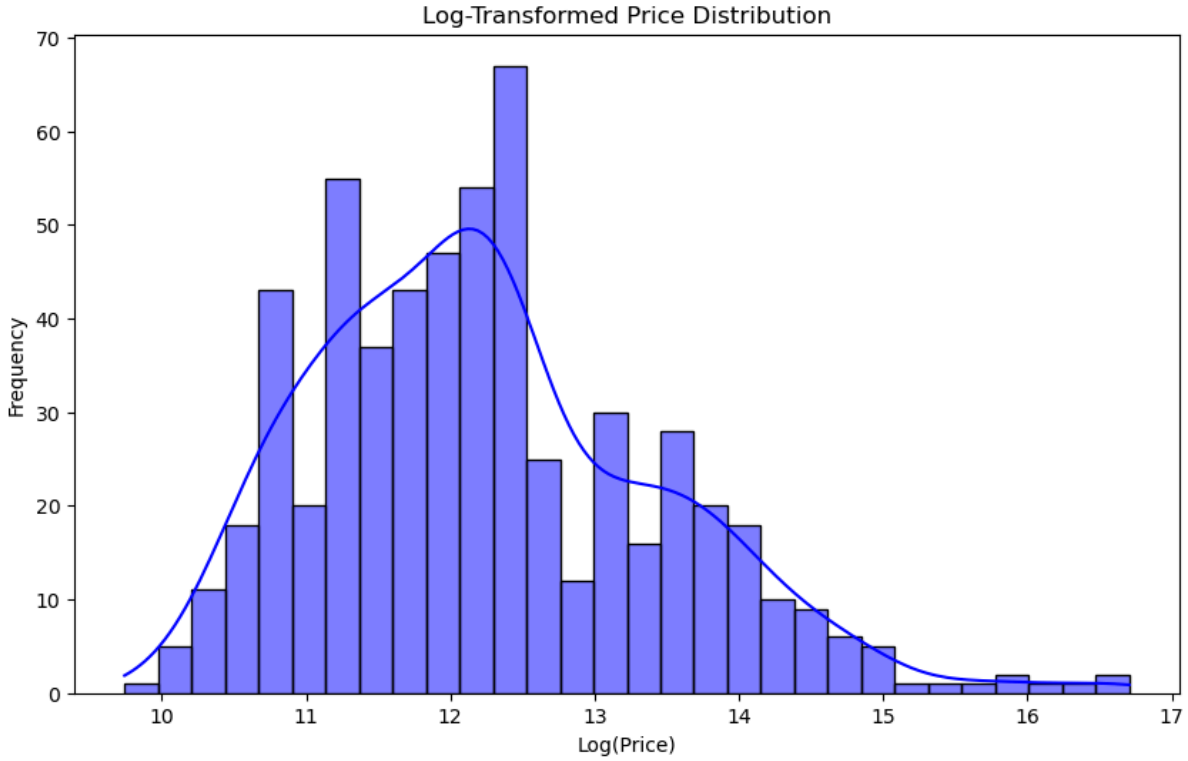


Figure 2: Price Distribution

When we initially started gathering our data, we made an effort to include houses of many different price ranges. Thus, it contained some properties on the extreme high-end of prices, with the maximum price being 18 million. When we made the histogram for the price, we observed that these outliers skewed our data and we could not interpret their distribution correctly. In order to address that, in Figure 2, we plotted the histogram against a log axis of price to approach the normal distribution. The distribution of log transformed prices has positive skewness, as we can see on the right tail. The max of the distribution is approximately in 12-13 log units, indicating that most of the house prices are concentrated around this range. This corresponds to prices from 160.000€ to 450.000€.
After 14 log units we have a small number of houses that are high priced. These houses are outliers that represent the more expensive properties and whereas they are few in number, they extend the tail of the distribution.
Before 10 log units we have few houses and that means that our data includes more mid and high priced houses.

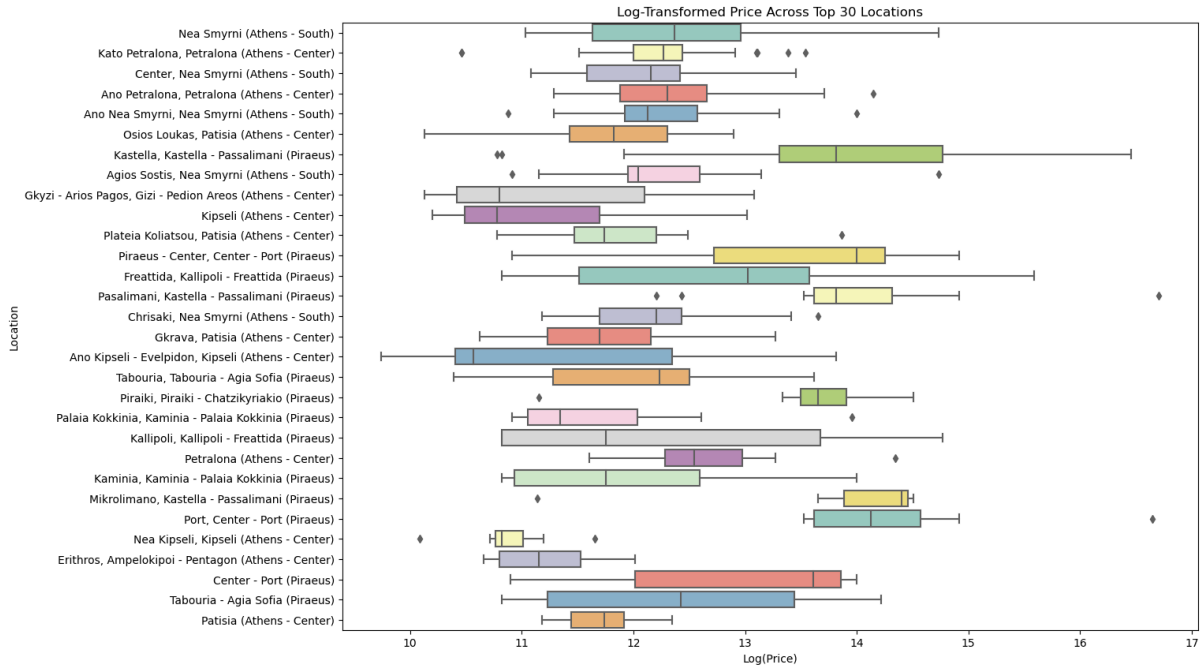


Figure 3: Price distribution per location

Another plot that is interesting is the boxplot of log price based on locations(Figure 3). Each of the boxes depicts the price distribution in a specific location. The log price distributions differ significantly across different locations. The max interquartile ranges belong to Kallipoli, Kallipoli - Freattida (Piraeus) showing no homogeneity in the property prices, and this means that we have variation in price. Also, the median is placed more left from center so we have negative asymmetry. In Kaminia - Palaia Kokkinia (Piraeus) the median is exactly in the center of the box and that means that we have the same prices along that location. In the locations, Nea Smyrni (Athens - South), Kato Petralona (Athens - Center), and Center, Nea Smyrni (Athens - South) have wide ranges of prices, indicating both affordable and premium properties in these areas. Kipseli (Athens - Center) and Patisia (Athens - Center) have smaller interquartile ranges and in this way there is more homogeneity in house pricing in these areas. We observe that many locations have outliers which are dots outside the whiskers. So, these properties are significantly more expensive than the majority in those areas. For example, we see that in Ano Petralona and in Ano Nea Smyrni we have outliers and that means that we have some houses at bigger prices compared to typical price ranges. Finally, locations with a higher median log price, such as Pasalimani, Kastella the majority of the properties are luxury.

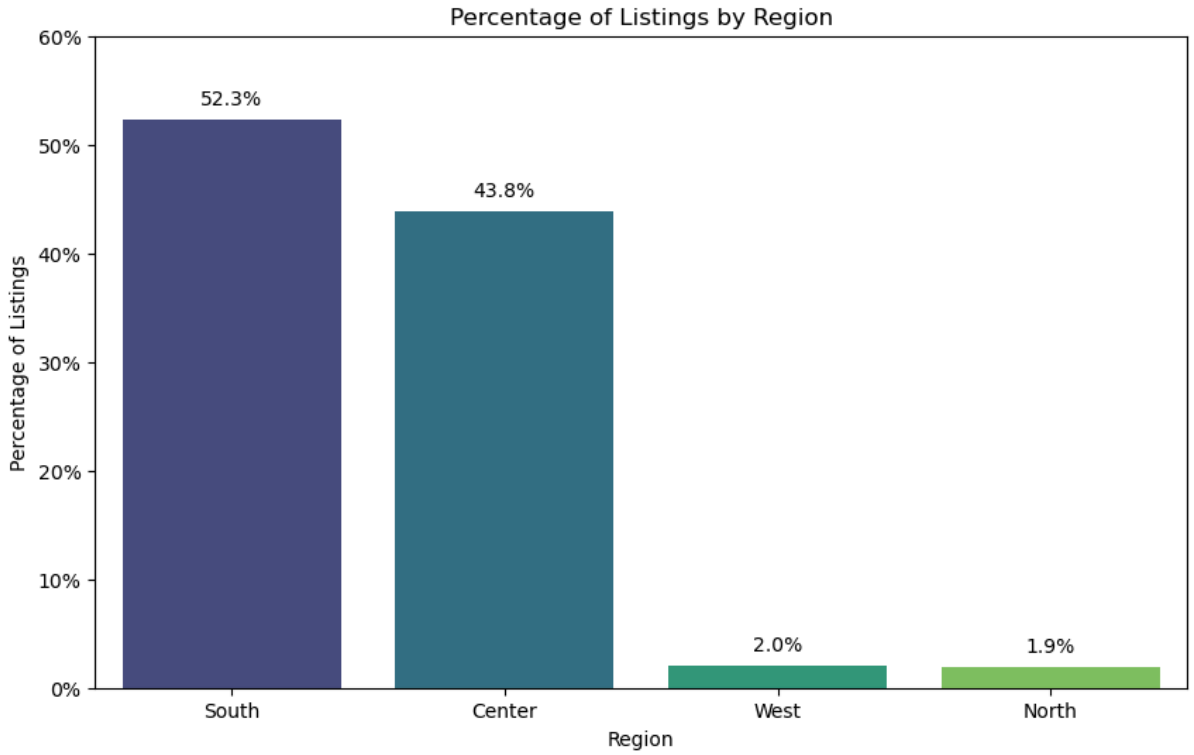


Figure 4: Count of listings per region (province)

In Figure 4, we made this bar plot to categorize each location in a specific region (e.g., South, Center, West, East and North) and also to see how many locations belong to each region. To begin with in the South, we have Nea Smyrni, Ano Nea Smyrni, Agios Sostis, Piraeus, Pasalimani, Kastella, Kallipoli, Kallithea, Charokopou. Next, in the West, we have Chaidari, Peristeri, Aigaleo, Korydallos, Nikaia, Galatsi, Poligono - Tourkovounia, Poligono. Moreover, in the Center, we have Petralona, Kato Petralona, Kolonaki, Exarchia, Goudi, Zografou, Gkyzi, Kipseli, Patisia, Ano Kipseli, Gkrava, Ampelokipoi, Panormou, Erithros, Pedion Areos, Ampelokipoi - Pentagon, Lofos Finopoulou, Platia Gkizi, Girokomeio. Finally, in the North, we have Marousi, Kifisia, Chalandri, Nea Ionia, Metamorfosi, Neo Psychiko, Paleo Psichiko, Nea Filothei, Drosopoulou. From the above plot we can observe that the most of the locations belong to the South with a percentage 52.3%. The 43.8% of the locations corresponding to the Center, only 2% in West and the fewest locations with 1.9% percentage belong to the North. We do not have locations in the East.

2.2.2. Image classification

Before price prediction we wanted to classify our images in order to conduct some exploratory data analysis. To achieve that we used a pretrained model from hugging face by using the transformers library. The model was Tater86/room-classifier-v1 which is reported to have an accuracy of about 90%. In our case, however, we found out that the model's accuracy was slightly decreased since many house ads had images of empty rooms with no furniture, thus leading to some false labeling.

Another reason that we did classification on images was we wanted to pass additional informational input about the images and their representation to the model. For example, many houses on the higher price tiers included images of pools. We hypothesized that including said information would increase the model's accuracy.
Next, we loaded each of the images in our directory and with the use of the model the room type is being predicted and the files are renamed like kitchen_1.jpg, living_room_2.jpg etc. The uniqueness of the predicted class is being ensured by the use of defaultdict. As the final step we made a function to count the room types. We see that most of the room types were living rooms, the least of them were laundry rooms.

```
entry room: 206
living room: 518
kitchen: 313
bathroom: 159
pool: 77
exterior: 324
hallway: 449
garage: 202
dining room: 252
aerial: 33
floorplan: 45
bedroom: 149
office: 22
laundry room: 15
```

Below, in Figure 5, we can see the visualization of the room types.

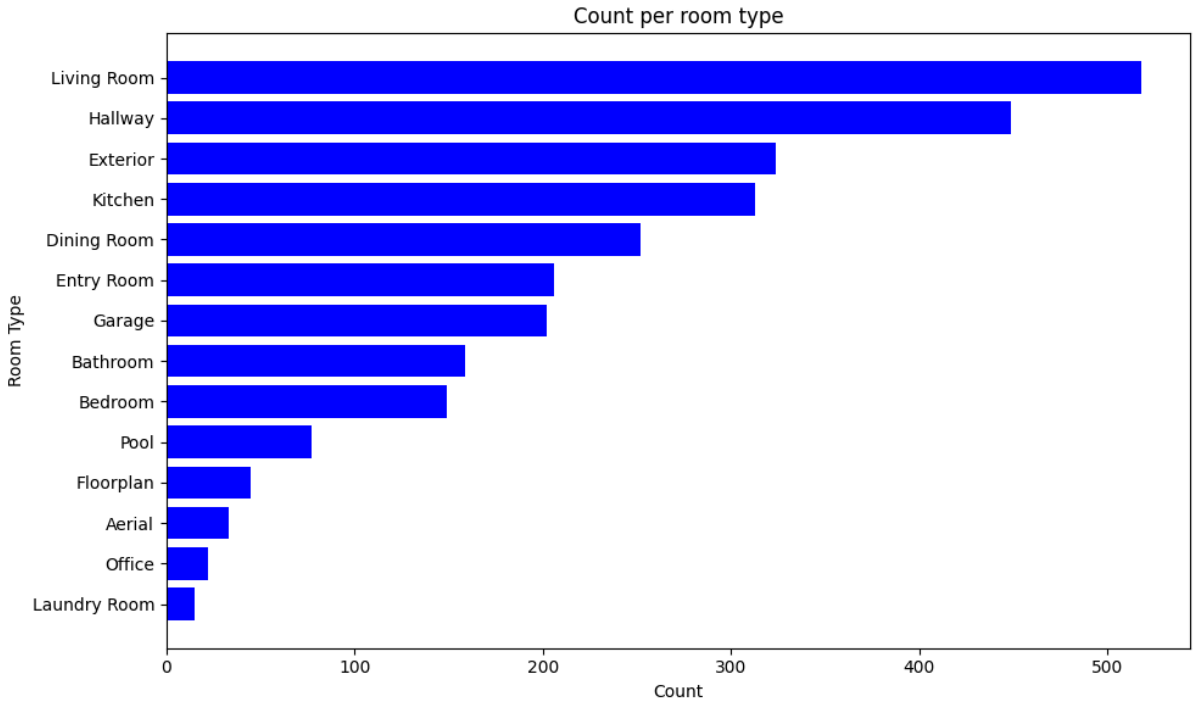


Figure 5: Barplot of room types

2.3. Data Preprocessing and Annotation Techniques

2.3.1. Data pre-processing steps

There were quite a few data pre-processing steps that we took, some general steps that applied to the whole dataset and some that were applied on the separate data sets that were derived from the maine dataset, in order to be able to handle the needs of each specific task in our project.

Main dataset pre-processing steps:
Missing Description: In quite a few of the ads that we collected, the description was missing. To make sure that our data followed a standard format, we changed the missing descriptions to “No description” to avoid possible errors that could be caused by it in the future of the project.

Price prediction pre-processing steps:
For the price prediction part of our project, there were a few additional changes that we applied on the original dataset in order for the data to be usable and well structured for the model training procedures.

Price conversion: In the dataset that we created from the scraper, the price feature was in the form of a string. We changed that to be a float, by removing the € symbol in order to be able to use the price in numerical processes and machine learning operations.

Adding Location: At the end of each description, to have a more complete description, we appended the location of the listing. This was done in order to process the location information alongside the description in a single input, thus limiting the model’s complexity.

Price bucketing: The original idea was to create a regression model with a single output that would represent the predicted price. In practice, however, the original model would not learn a lot from our train data and would predict somewhere close to the mean price of the dataset, with a small deviation according to the house’s actual price. A lot of different ideas were tested to amend this, like standardizing or normalizing the price, but in the end, the problem persisted. After some thought, we decided to switch our approach from a regression model to a classification model, which would classify a house into a certain price range.

For us, to run classification, we had to change the continuous price values into categorical. We created 5 buckets of the prices, essentially for the following price categories:

Very Low: 0 - 75.000 €
Low: 75.000 - 140.000 €
Medium: 140.000 - 230.000 €
High: 230.000 - 580.000 €
Very High: 580.000 - 18.000.000 €

These buckets were created by applying qcut on our dataset, in order to ensure a balanced number of observations in each category (around 120 in each bucket).

Image standardization: As the images that we scraped were in various resolutions and sizes, we had to standardize them to 512x512 pixels in order to have the same dimension for the convolutional neural network (CNN) that was used later on.

2.3.2. Data Augmentation

House description:

In advance of running the model for fine-tuning, we thought that it would be beneficial to perform some type of data augmentation technique and increase the listings that were contained in our original dataset (600 listings). The data augmentation technique that we found fitting was paraphrasing of the ad descriptions. The script that we created, interacted with the Groq API and Llama3-groq-70b model. For each listing in our dataset, the paraphrase generated 3 additional alternatives of the listing. So from 600 ads we got to around 2,500 ads.
In the beginning of the process, we used as input into the Groq API the description of the ad. We made sure that the paraphrase would return 3 different versions of the description by rephrasing and using synonyms of certain words or sequences of words. At the end, the new description, together with the originals were saved in a new CSV file that was later used in the model training process.
In order to generate the 3 descriptions we used prompt engineering. After trying a variety of prompts, we concluded that the one that would result in the best output from the LLM is the following:

"You are a real estate ad paraphraser AI tool. Paraphrase the ad descriptions given, maintaining all information.
Use a different style of language.
Mix the information sequence.
Do not add or remove any information.
Return only the rephrased ad, no introductory sentences."

Composite images:

One of the first considerations for the price prediction model was to figure out how the images would be fed to the input layer of our cnn. Creating multiple inputs in order to feed each image one by one would overly complicate our model as it would introduce a lot of parameters. After some research, we followed a recommendation of combining different images into a single composite image. That way, we could feed multiple images to the model with a single input, thus limiting the complexity.

Since we have downloaded the listings' images in a local directory, our goal was then to create collages of images from each folder. At this point, we also applied augmentation on images with the use of the Albumentations library to enhance the variety of images and achieve diversity to the final composites. As an added bonus, our training dataset was enhanced and effectively tripled in size. For the image augmentation, we applied horizontal flipping with probability 0.5, random rotation up to ±20 degrees, adjusted both the brightness and contrast of the image with probability equal to 0.2 and finally scaled the image within a limit of 80-120%.

Our composite images include four images in a 2x2 grid. Each of these images was resized in 256x256 pixels and the final composite is an image with 512x512 pixels. For each folder, we created three different composites. The first composite is a random selection of four images from the folder. The second composite is again a random selection of four images which are augmented from the augmentation pipeline. The third composite image is the same procedure with the second composite. With random selection we select four images at random from those sets that are available in the folder. In case the folder contains less than four images, the last image is repeated so the total number of images is equal to four. With the conclusion of this process, each folder contained three composite images that are all slightly different from each other due to the random image selection and augmentation.

The pictures below are an example of the composite images created for a single property listing:

2.3.3. Ad Feature Extraction Process:

Here, our main focus is on extracting important features from the real estate ads, which play a crucial role for both the description enhancement and the price prediction tasks. These key features help to improve the model's understanding on the specific dataset that we are working with.

The process is done with Python and takes advantage of quite a few libraries such as “re” for finding regular expressions, “groq” for API access and “pandas” for data manipulation.
The main part of that whole process is the function “ad_description_features”. It utilizes the model “llama3-groq-70b-8192-tool-use-preview” by sending requests through Groq API. We chose to use that model, as it is fine-tuned for instructions and for function calling. Another model that we had previously tried for that purpose (llama3.1.) would hallucinate, giving us back results not in the correct form that we had prompted for, or with completely wrong information. That happened because the prompt was too complex for a non fine-tuned model and it could not handle it correctly.

The script utilizes prompt engineering in order to make the model act as a real estate assistant and to extract 4 specific features that we thought to be important and valued by the customer to know about:

1. **Public Transport**
2. **Amenities**
3. **Layout**
4. **House type**

The features that we care about and the prompt, that after trial and error, we found to be the most effective are the following:

> ***“You are a helpful real estate assistant. You take real estate descriptions and check if they contain the features below:***
> ***1) Public transport stations (metro,bus,etc)***
> ***2)Amenities (for example: supermarkets, parks, schools, universities,Shopping centers and grocery stores, parking, A/C, not rooms like kitchen or bathroom)***
> ***3)Layout (how many and what rooms it has, size is not considered a layout)***
> ***4)House type (apartment, full house, building, etc)***
> ***First answer if the features are present with yes/no and then specify why.***
> ***Don't assume information not specified in the description.***
> ***Don't write information not asked above.”***

Then we processed the ad descriptions in batches by reading the Excel file with the listings descriptions, then going through each listing and extracting its features. Lastly, the results are organized and stored in a CSV file that is named “ads_groq_output_test.csv”.

Finally, we checked the outputs in the CSV file through random sampling of ⅓ of them. We checked that the output contained correct and accurate information while having a correct

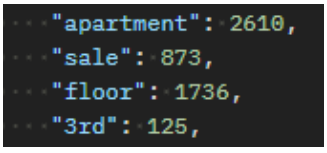
list-like form. Wrong outputs were cut from the final CSV file in order for the model to perform closer to our expectations after the fine-tuning .

The output data of that process, were later on used in the creation of the dataset that was utilized for the process of fine tuning.

2.3.4. Description Text pre-processing

In order to use the description texts as inputs in our price prediction model, they had to be transformed into a suitable format. The scrapped descriptions were a string type, but in order to be processed by the neural network, they needed to be converted into vectors. One such method that produces vectors from texts is tokenization. To start the process, a tokenizer is initialized taking as input the vocabulary length, meaning the number of different tokens we have. In our case, we selected a vocabulary length of 10000 words, which was sufficient for our training data.
Before fitting the tokenizer, we applied some further cleaning in our descriptions. In any free text there exist words (or tokens in our case) that do not carry any substantial information and may increase the noise in the text. There is an extensive list of these stopwords, which is different for any given language. We added to this list certain tokens we found in our data that appeared frequently but had no semantic importance (like parentheses). In order to have more effective representations of the descriptions, we decided to remove these so called stopwords from the training data.
After removing the stopwords, the tokenizer was fitted on our data and each description was transformed to be a vector of numbers. Each number corresponds to a single token, since the tokenizer acts like a key-value pair dictionary. The resulting sequences were then padded, in order to have a standard input size for the model, regardless of description length. The padding length was selected to be equal to the maximum size within our description token sequences.

Tokenizer snippet:



3. Machine Learning Architectures and Methodologies

3.1. LLM Methodology

Feature Extraction Model Fine-Tuning

In the beginning of our assignment, for the House description and feature extraction task, although we would get really strong results using a pre-trained Sequential model, we noticed that quite few of the results would have a common issue known as hallucination. The model would give us irrelevant information, that we did not ask for or that was not even included in the original text. Additionally, we notice that in our dataset there would be certain common features across all the ads, like specific terminology and structure of the descriptions, that the model could misinterpret and generalize the information out of it wrongly. We thought that the original model could have been trained on data that had different characteristics, so fine-tuning of the model could help us overcome the problem of hallucinations and lead us to even better results.

For the fine-tuning, the model that we utilized was a 4-bit quantized version of the Llama-3.1 8b parameter model in order to be more efficient with memory usage and to get faster results. We chose that, as it is trained on a large dataset so it is well suited for our goal of fine-tuning.
The two main libraries that were used during that process are the “unsloth”(to load the model) and “torch”.

Some technical details for the fine-tuning process:

- Number of GPUs Used: 1
- Number of training samples: 2548 & number of epochs: 1
- Batch size per device: 2 with Gradient accumulation steps: 4
- Gradient Accumulation steps: 4
- Total batch size: 8 with 318 total training steps
- Number of trainable parameters 41,943,040

The duration of the training process was 3,548.8053 seconds or around 59.15 minutes, with the max memory usage to be at 8.205 GB. In more detail, the memory reserved for the training was at 2.221 GB, that is about 15.06% of the maximum available GPU memory. That shows us the efficiency of the fine-tuning on managing the limited resources that we had available.

We set the model to use LoRA (Low-Rank Adaptation) adapters that can help with doing the process of fine-tuning with fewer resources (like 1 GPU in our case) by utilizing only a small percentage of the model’s parameters (1-10%), without sacrificing the effectiveness of fine-tuning.
In order for the model to be able to handle different sequences of lengths, we used Roraty Position Embeddings (RoPE) Scaling. That way, we don't have to make adjustments to the parameter “max_seq_length” every time.

To achieve even better memory usage optimization during the fine-tuning, we took advantage from the “unsloth” library, the gradient checkpointing. Consequently, we can use larger batch sizes and context lengths, as it lowers the computational cost of training long sequences by decreasing memory overhead.
Then, we utilized from the Huggingface TRL's library, the “SFTTrainer” in order to perform the fine-tune of the model on our custom formatted dataset. We did 60 steps (short duration) in order to get faster results with small batch sizes.
Lastly, after the process of fine-tuning has been completed successfully, the model that was used was saved not only locally but on Hugging Face Hub as well to make sure it is accessible and available for future deployments on real time applications.

The dataset that was used (chatml_fine_tuning_data.csv), was transformed in a format that when provided as input, the model will be trained properly. The dataset was formatted as a CSV file with 3 columns, “System”, “User”, “Assistant”. The “System” column provides a prompt that will steer the model, “User” is the description of the ad and the “Assistant” is the text based output the model is expected by us to return.

User and Image Description Comparison and Combination

Our main focus for this implementation was to utilize the latent information in the listings’ images in order to compensate for the user’s description inaccuracies. For that we used image to text conversion methods and a description summarizer. We started by taking the images as input in an image-to-text model and after that, we used the descriptions that were generated by the image-to-text model and fed them into a summarization model in order to create a complete description of the house.

For the image-to-text description generation the model used was “llava-v1.5-7b”, which is a pre-trained LM for image processing tasks. The model that we first tried was the “MiniCPM-Llama3-V-2_5-int4”, but later on we replaced it, as we found that the llava model is available for free from the Groq API, meaning that its inferencing is ridiculously fast. For one image, the description generation time was reduced from 10 seconds, to just 1 second by using Groq.

Using the prompt **"Describe the room in the picture and say what room type this is. You can also comment on lighting, floors and furniture. Do not mention personal use devices like laptops, computers, smartphones etc."** we created a description for each picture that was provided on the listing.

In the next step, we took all the descriptions from the images that were generated and passed them in the summarization model in order to create a final total summary of the house listing. The model that was used here was the Llama-3-Groq-70B-Tool-Use through the Groq API.

Carrying on, by taking advantage of the model “Llama-3-Groq-70B-Tool-Use”, we focused on

the extraction and standardization of key features from the descriptions. We saved the features on a standard format list, so they can be easily compared and that list includes information for at least one of the following features: lighting, flooring. The detection of the features that we were interested in happened by using regex patterns.

After extracting the features from the user description, we repeated this process for the generated description from the listing's images. In order to create a meaningful comparison between the two, we needed to vectorize them to calculate a similarity metric. We created embeddings by using the SentenceTransformer model 'all-MiniLM-L6-v2'. After we had the embeddings, we calculated the cosine similarity in order to be able to compare their similarity of the descriptions for each of the two extracted features, lighting and flooring.

As an added feature, we created an implementation that takes the user inputted description and the model generated one and combines them, thus creating a new enhanced description. In order to create a better description that contains information from both, we used the Llama-3-Grow-70B-Tool-Use. After a few trials the prompt that was giving us the best results is the following:
"You will be given descriptions from house rooms. Combine the two descriptions into one better paragraph that has all the information needed for a real estate ad. Please mention area, price, contact and all necessary information. Do not mention lack of description for any room or feature. Do not mention personal use devices that when a house is sold they are not present like TVs or computers. Provide the ad directly without any introductory phrases.Please use a simple style of natural language."

3.2. Price Prediction Methodology

For the price estimation model, our first consideration was to develop an ensemble model. Each sub-model would have a different input, for example one model for images and a separate one for the descriptions. The classification result would then be derived by majority voting, with each model's vote weighted by its accuracy. In practice, however, the separate models we implemented struggled to score within an acceptable range (usually between 30 to 35% accuracy). This indicated to us that the price classification problem was complex and would need larger and more sophisticated models in order to tackle. That said, we started the implementation of a multimodal model that would combine the various inputs in order to produce a classification prediction. The results of the multimodal model proved more promising than our efforts with the ensemble models.
More specifically, we built a model that has three different branches and can combine different input data and then merge them to make the final prediction. Even though it is more complex than the previously tested ensemble models, it has 6 million parameters , or 23 MB of memory, so it is relatively lightweight as well. This was instrumental for the training phase, since it meant we could load the model and train it on the limited resources we had available.
In the following sections we will discuss in depth the model architecture.

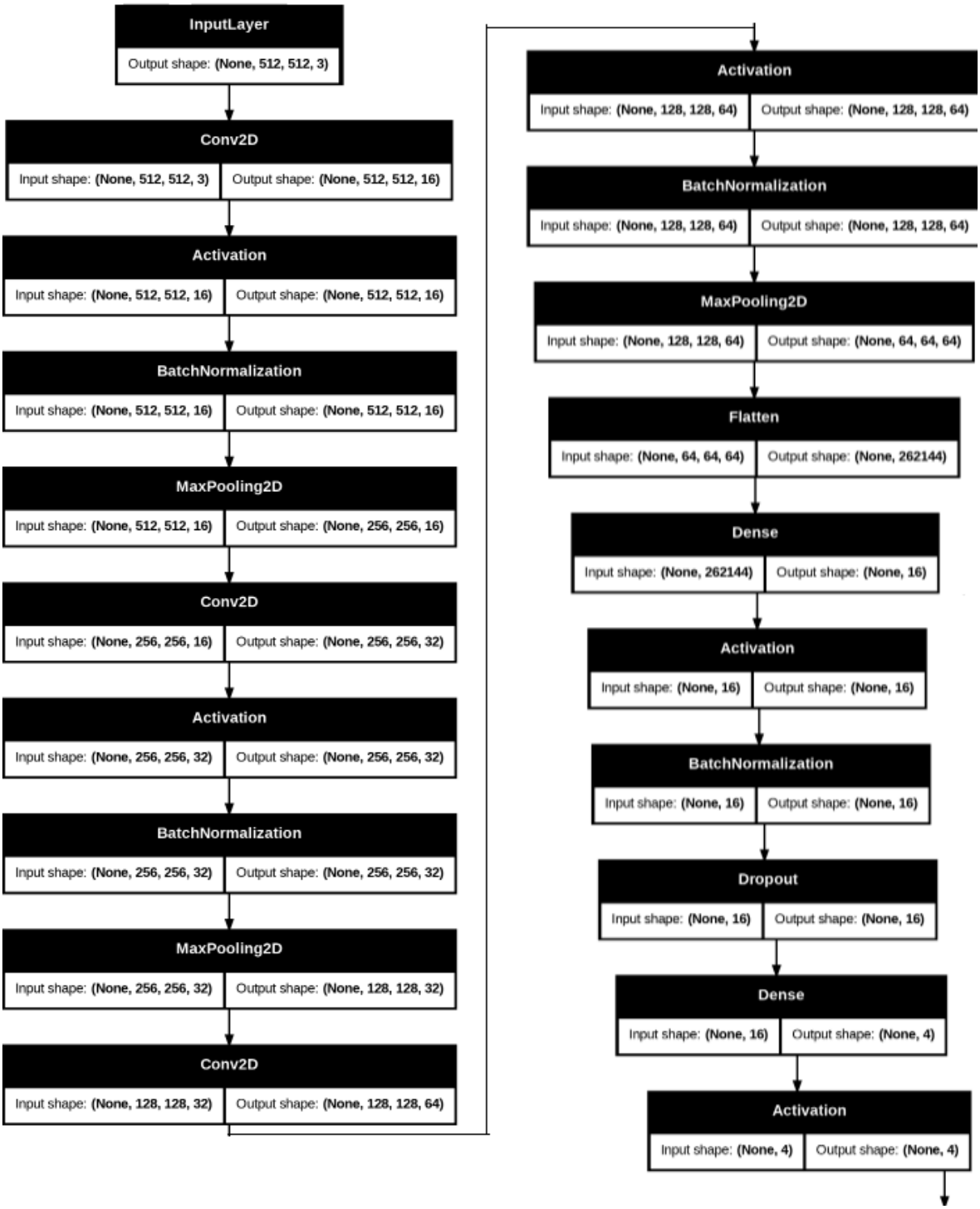
Image Processing Branch (CNN)

The network architecture for the image processing branch is that of a Convolutional Neural Network (CNN). A CNN is a specialized form of feed-forward neural network that includes regularization and automatically learns features through the optimization of filters (or kernels). This deep learning model is used across a variety of data types, such as text, images, and audio, to make predictions and perform tasks.

The architecture of this branch was based on a model proposed by user amir22010 on kaggle. This CNN proved very adaptable to our scenario and scored better accuracy results when compared to some other pretrained models we tested, such as the ResNet50 imagenet model. In addition, it required fewer trainable parameters, so it expedited the training process, particularly when taking into account the limited resources we had available.

This branch is fed a composite image of size 512×512×3 width, height, and RGB color channels represented in that order, respectively. Each convolutional layer applies a suite of variously sized filters on the input image to better extract features. These usually start from simple features that include edges and textures to complex features involving shapes and other objects. After each convolutional layer, we apply an activation layer, with ReLU as the activation function in order to introduce non-linearity. Next, we follow it by a Batch Normalization layer for stabilizing the process of learning and accelerating convergence. We finally apply a pooling layer. The pooling operation involves sliding a two-dimensional filter over each channel of the feature map and summarizing the features lying within the region covered by the filter. Max pooling reduces the spatial dimensions of an image, enabling focus on the most striking features within that image and doing so diminishes a lot of computational complexity.

After concluding with the convolutional filters, we implemented some flatten and dense layers meaning that the final convolutional output is flattened into a one-dimensional vector that goes through some dense layers. These layers along with dropout are useful for regularization to learn abstract representations from image data. Dropout was added in order to mitigate the effects of overfitting.

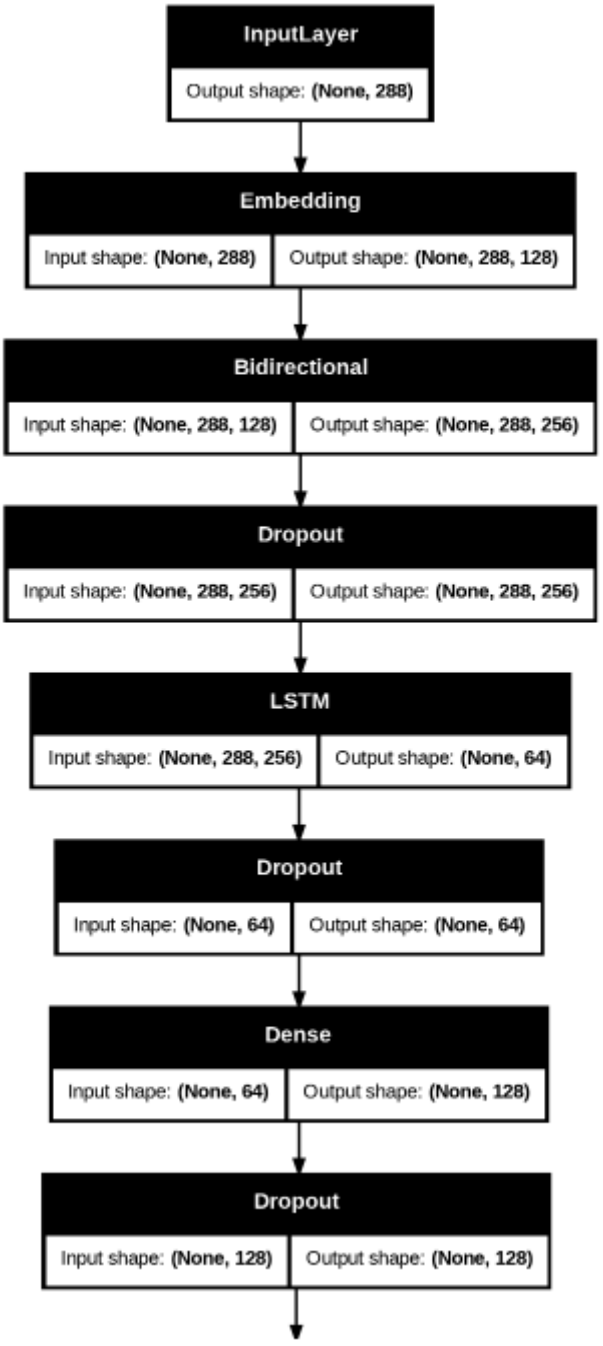


Text Processing Branch (RNN)

For the text processing branch of our network, we implemented a Recurrent Neural Network (RNN). RNN is a deep learning model that is trained to process and convert a sequential data input into a specific sequential data output. Sequential data is data—such as words,

sentences, or time-series data—where sequential components interrelate based on complex semantics and syntax rules. The key feature of RNNs is their ability to maintain a "memory" of previous inputs in a sequence, which allows them to capture temporal dependencies.

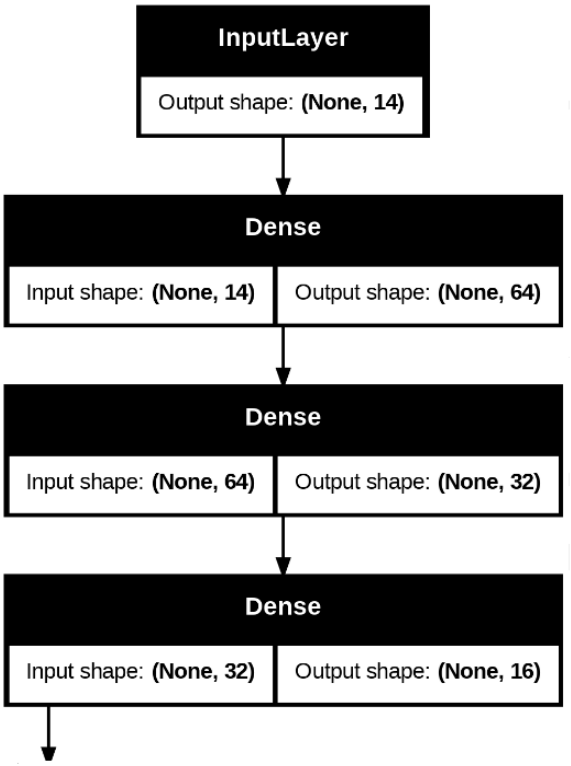
The input layer of the text processing branch is fed with the processed descriptions, which are now formatted as vectors through the process described in the data preprocessing section. Next, we utilize an Embedding Layer in order to convert the integer-encoded tokens into dense vectors, reducing dimensionality to minimize the trainable parameters. Following that, we implement a Bidirectional Layer, which runs the input data in two directions: one from past to future and one from future to past. This is useful when context from both directions can help predictions, such as when knowing words that come later can help interpret earlier words. The output size is doubled compared to the input size because of the forward and backward pass. The next notable layer is an LSTM (Long Short-Term Memory) which is a type of recurrent neural network (RNN) that is designed to remember long-term dependencies. The LSTM cell can capture information from earlier parts of the sequence and avoid problems like vanishing gradients. Finally, we implemented a fully connected layer which allows for non-linear transformations of the input data, helping the model to make more complex predictions. We also add Dropout Layers throughout in order to prevent overfitting.



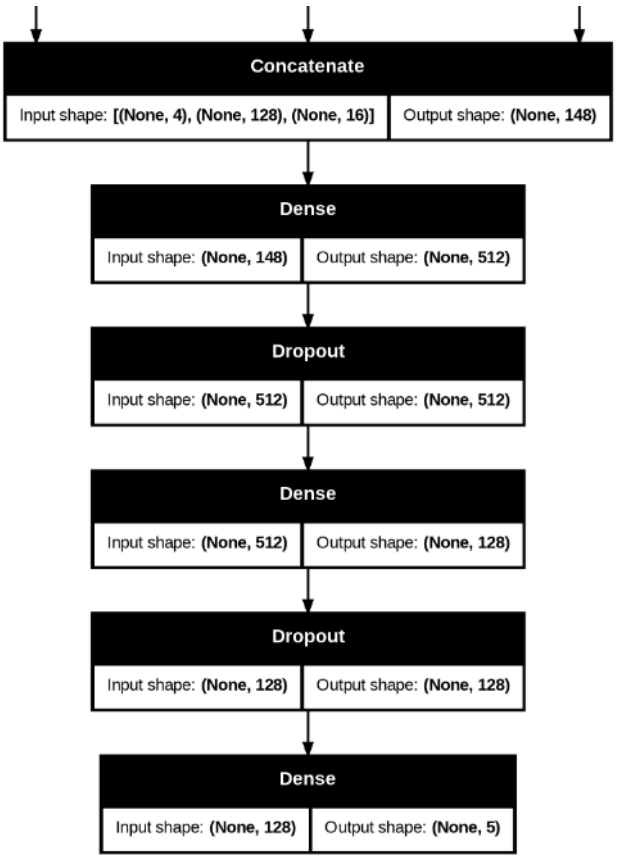
Room Label Branch

This is by far the simplest branch in terms of parameters, with only 2KB of trainable parameters. The room label branch is a feed-forward network. The input is a single vector of 14 dimensions, equal to the number of room labels at our disposal. This vector was produced by using a Multi Label Binarizer, which transforms a set of labels for a multi-label classification problem into a binary format. The vector represents the presence, or lack thereof, of each room label in a binary format. Then, the input is fed through a series of fully

connected (dense) layers, each with a successive reduction in the number of neurons which are 64, 32, and 16. Successive reduction in dimensionality learns key interactions among the features gradually. ReLU activation functions are used in each of these dense layers to introduce non-linearity so that the model can learn complex patterns in data.



As a next step, we join the resulting outputs from the three branches, image, text, and room labels. In order to combine the various outputs into a single input, we utilized the Concatenate Layer which returns a single tensor to be processed as input by the rest of the network. The joined vector is going to represent the whole of the property as it comprises all the visual, and textual features. The joined vector is then sent through some dense layers so that additional learning can take place. They integrate features from all branches and bring about further refinement of the final predictions. In order to avoid overfitting, dropout is used between these dense layers and this ensures that the model does not depend too much on certain neurons. The final layer of this model is a dense layer with the softmax activation function that gives the probability distribution in the five categories of prices. The one with a higher probability will be the model's prediction.



Finally, the model was compiled using the Adam optimizer for the training process. Adam dynamically adapts the learning rate based on the first and second moments of the gradients. This gives the possibility of faster convergence to the best performance. It uses the sparse categorical cross entropy loss function, which is applicable for multi-class classification problems in which the output should be a single integer class label and as metric of evaluation we calculate accuracy, considering that it looks at the ratio between those that were correctly classified against the set of all the predictions. The resulting model had around 6 million trainable parameters and a size of 23 MB.

Next, we will focus on the training process of our model. The model was given the three input vectors, as well as the target variable (price bin). At each training step, 20% of the input data was kept aside to use as a validation dataset and we measured the prediction accuracy in it. The validation accuracy is more helpful than the training one, because it helps us visualize how our model generalizes over unseen data.

Since we could not load all the training data simultaneously for each training step due to RAM limitations, we loaded them in batches of 48. We also decided to train the model for 400 epochs, meaning 400 different passes over our training data. In actuality, the training process concluded before reaching 400 epochs thanks to the callbacks used. In order to optimize this process, we utilized two keras callbacks, ModelCheckpoint and EarlyStopping. With ModelCheckpoint, we made sure to save the best model, in terms of validation accuracy, to a keras file, in order to access it at a later date. The EarlyStopping callback was implemented in order to stop the training process if the model did not improve over consecutive epochs. This callback would monitor the validation accuracy in each epoch and

terminate the training process after 50 consecutive non-improvements from the best reported one.

```
[ ] history = model.fit(
        [images, text_padded, room_labels_encoded],
        y,
        epochs=400,
        batch_size=48,
        validation_split=0.2,
        callbacks=[early_stopping, checkpoint]
    )
```

```
Epoch 1/400
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 577ms/step - accuracy: 0.2270 - loss: 1.6193
Epoch 1: val_accuracy improved from -inf to 0.27358, saving model to /content/drive/My Drive/ML project/Kostas/best_model3.keras
27/27 ━━━━━━━━━━━━━━━━━━━━ 32s 772ms/step - accuracy: 0.2275 - loss: 1.6189 - val_accuracy: 0.2736 - val_loss: 1.5991
Epoch 2/400
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 519ms/step - accuracy: 0.3053 - loss: 1.5562
Epoch 2: val_accuracy did not improve from 0.27358
27/27 ━━━━━━━━━━━━━━━━━━━━ 15s 558ms/step - accuracy: 0.3055 - loss: 1.5558 - val_accuracy: 0.2358 - val_loss: 1.6238
Epoch 3/400
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 526ms/step - accuracy: 0.3353 - loss: 1.5006
Epoch 3: val_accuracy did not improve from 0.27358
27/27 ━━━━━━━━━━━━━━━━━━━━ 15s 564ms/step - accuracy: 0.3354 - loss: 1.5007 - val_accuracy: 0.2296 - val_loss: 1.7002
```

We commenced the training process using the setup above. After around 200 epochs, the EarlyStopping callback was activated, giving us the final model weights. The validation accuracy over the whole training process is visualized below:

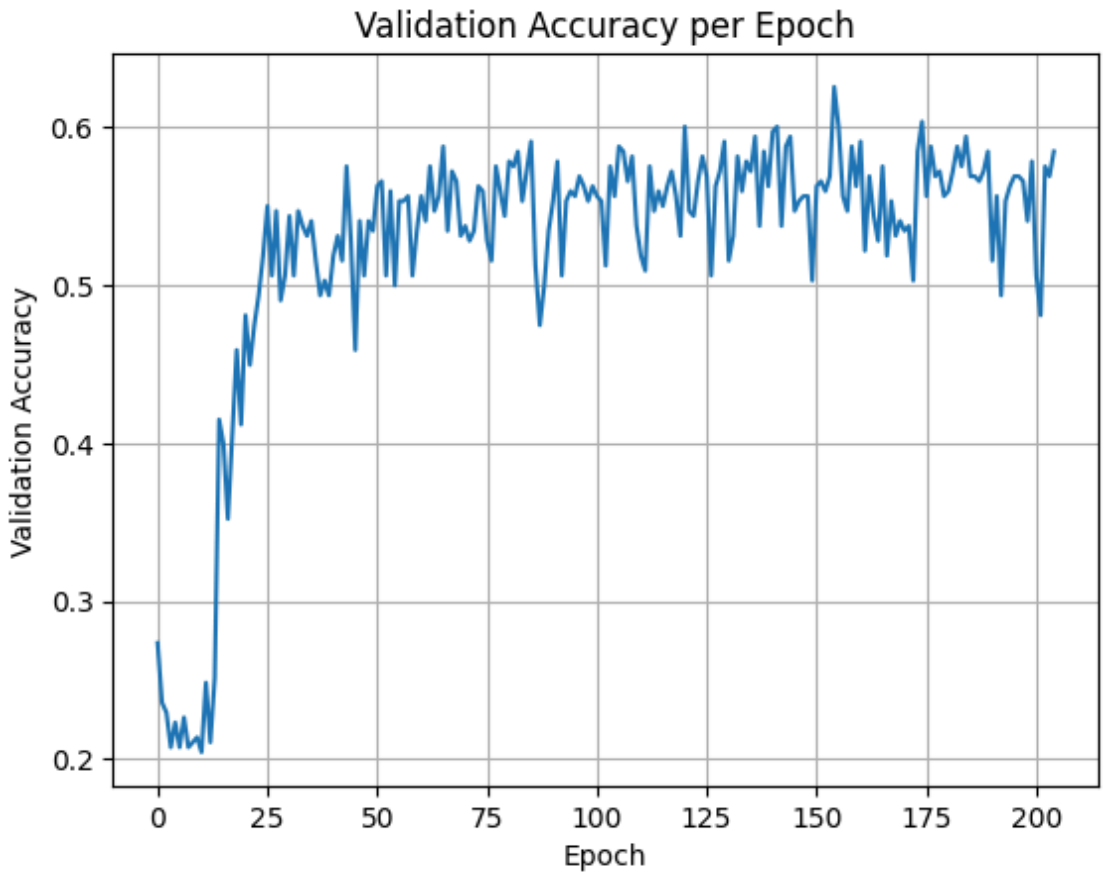


Figure 6: Validation accuracy across training epochs

The model's accuracy starts at about 20%, equivalent to random chance and increases to above 50% in about 25 training epochs. From there, it fluctuated between 50 and 60% and, after about 200 epochs, the EarlyStopping callback kicked in, giving us the best model at about 61% validation accuracy.

4. Experimental Setup and Tools Used

Tools

All the steps of the project were implemented by the use of the **Python programming language**. Python provides a vast suite of libraries that were essential to our work through every stage, from the data gathering with the web scraper, to the data preprocessing, analysis and visualizations and lastly to the models' architecture, the training and fine-tuning.

For the model training and LLM fine-tuning in particular, we utilize the **Google Colab** platform, which supports Jupyter Notebooks and provides free access to computer resources essential for these processes. The T4 GPU runtime was particularly useful, since it provided 15 GB of VRAM. Utilizing the parallel processing capabilities of the GPU, the training process could be completed much faster. The only limitation was that access to this GPU runtime was time-restricted, so we often needed to run the notebook from multiple profiles in order to finalize it.

Libraries

- Selenium: Was utilized in the development of the scraper that the creation of our dataset was depended on. It would go through the links of houses ads and help to get key features.
- TensorFlow.keras: They helped us in the development of models for image processing, text classification (house feature detection) and fine-tuning.
- Sklearn (Scikit-learn): Was used in the majority of machine learning operations, such as classification and regression, especially on feature detection and price prediction.
- Numpy: Was used mainly on doing different numerical transformations in the datasets.
- Pandas: Played a crucial role in cleaning and shaping the raw data that we had, so we can take advantage of them to the fullest in our project.
- Transformers: That library was used in the process of the fine-tuning for our model. With its superior NLP capabilities, it helped us alot for the tasks of feature detection and if text paraphrasing.
- Torch (PyTorch): It was employed in combination with Transformers to allow us and use model training techniques more efficiently.
- Groq: It played a crucial role in allowing us to call different language models throughout our project (like Llama3-groq-70b) for our objectives like paraphrasing descriptions, highlighting features based on the descriptions.
- PIL (Python Imaging Library): It helped us in resizing and standardizing the dimensions of the images we had from the house listings.
- User Agent: Added some randomness like random movements and scrolling during the scraping process, so we were not flagged by CAPTCHA as bot.
- Other miscellaneous Python libraries are: Skimage.io, OS, Locale, Argparse, Datetime, MatPlotLib, Seaborn.

5. Results & Analysis

5.1. Results & Quantitative Analysis

For the task of the feature extraction, we reviewed a random sample of results to ensure that everything was adequate to our expectations. Here we will present the results that we got from a house listing.

Listing Description from site:

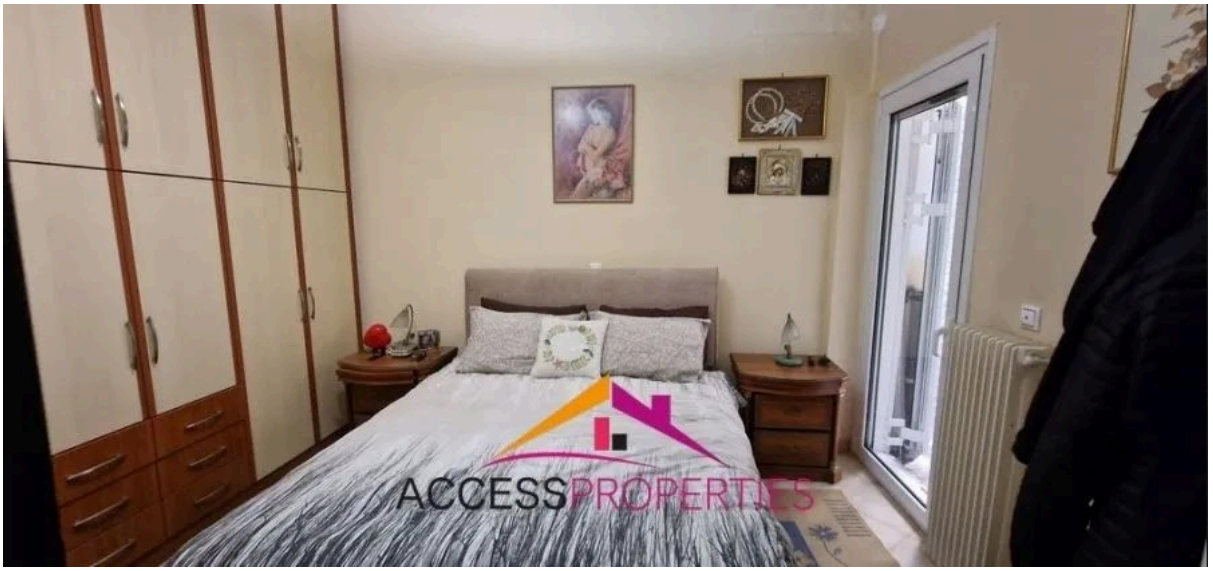
“Galatsi, Aktimonon, Floor Apartment For Sale, 115 sq.m., Property Status: Amazing, Floor: Ground floor, 1 Level(s), 3 Bedrooms 1 Kitchen(s), 1 Bathroom(s), Heating: Personal - Petrol, Building Year: 1980, Renovated in: 2004, Energy Certificate: F, Floor type: Marble, Type of door frames: Aluminum, Features: Security door, Storage room, Night stream, Double Glazed Windows, Window Screens, Balcony Cover, Balconies, Pets Allowed, Price Negotiable, Luxury, Airy, Roadside, Bright, AirConditioning, Solar water system, Boiler, Painted, Price: 194.000€. Access Properties, Tel: +30.210****, +30.69********, email: info@accessproperties.gr"***

House description and feature comparison Results Check

Below, we provide an example that illustrates how our model performs in the task of house description and feature comparison. This example highlights the process in which the model generates room-specific descriptions from the provided images, aggregates them into a complete house summary, and then compares this generated description with the original listing. Through this comparison, discrepancies are identified, allowing us to enhance the overall quality and accuracy of the listing’s description.

For the first step of the image-to-text description generation, we feed the model with three images and for each, we get the respectively descriptions

Image 1:



Description Generated:

”The room depicted in the image is a bedroom, typically used for sleeping and personal relaxation. It features a bed with a modern design, indicating a contemporary setting. The lighting appears to be ambient and soft, likely provided by ceiling fixtures or wall-mounted lights, which are not visible but suggested by the overall illumination of the space. The floor is covered with a patterned carpet that adds warmth to the room and complements the wooden furniture. The presence of a nightstand and a wardrobe suggests functionality and storage space typical of a bedroom. The artwork and personal items on the nightstand add a lived-in feel, suggesting that this is a private space within a residence.”

Image 2:

Description Generated:
”The room in the picture is a bathroom, as indicated by the presence of a toilet and a shower. The lighting appears to be artificial, likely from ceiling fixtures, which gives the space a bright and clean appearance. The floor is tiled, which is common for bathrooms due to its water-resistant properties. The furniture includes a vanity with a sink and mirror, typical for personal grooming and storage. The open door suggests that the bathroom is part of a larger living space, possibly an apartment or house. The watermark indicates that the image may be from a property listing or a real estate advertisement.”

Image 3:

Description Generated:
”The room in the picture is a living room, typically used for relaxation and socializing. The lighting appears to be natural, supplemented by ceiling fixtures, which provide a warm ambiance. The floor is not fully visible, but it seems to be of a light color, possibly tiled or laminated, which complements the room's decor. The furniture includes a sofa, armchairs, and a coffee table, arranged to facilitate conversation and comfort. The presence of a television suggests that this space is also used for entertainment. The overall setting is one that is designed for comfort and leisure, common in residential homes.”

On to the next step for the final total summary of the descriptions from the previous images, here are the resulted description that was generated:

“The house features a bedroom with a large bed, nightstands, lamps, a built-in wardrobe, and artwork, showcasing a comfortable and well-maintained space with wooden flooring and carpet for a cozy feel. A modern bathroom with a vibrant color scheme, natural lighting, and practical furniture such as a vanity and towel rack is also present. Lastly, a living room with comfortable seating, a central coffee table, and decorative items, illuminated by natural and artificial light, completes the house setup.”

Next as we move on to the extraction of features and standardization the returned results had the following form:

Site Description:
lighting: bright
flooring: marble

Image Description:
lighting: ample
flooring: wood, marble

The output for the similarities of the features has the following form:
lighting_similarity: 0.31187933683395386,
flooring_similarity: 0.5414676666259766

Meaning that the two descriptions disagree on lightning, but they are somewhat similar in flooring.

Feature Extraction Results Check

For the feature extraction portion of the project, we inserted the description above to our fine-tuned LLM. Below is the response of the model:

1. Public transport stations: No
2. Amenities: No
3. Layout: Yes, 3 Bedrooms, 1 Kitchen, 1 Bathroom
4. House type: Yes, apartment

As we can observe, the description does not indeed contain information about public transport and other amenities. It does however provide the house layout and house description. To test whether the model can successfully identify the two missing features, we inserted the description below which contains them:

GYZI - ARIOS PAGOS RENOVATED APARTMENT FOR SALE, total area 75 sq.m., on the third (3rd) floor of an apartment building. Layout: -1 living room -1 kitchen -1 entry room -2 rooms (1 can be used as an office) -1 communication corridor -1 bathroom. It has: -tile-wood floor -aluminum frames with double glazing -oven with stove, fridge -safety door -air conditioning (2a/c) -screens bathroom side -awnings -low utilities. It is frontage and bright during the day. It is located a 10-minute walk from AMPELOKIPI METRO station, near the Evelpidon Courts, Hospitals and a main road with frequent public transportation. Ideal for cohabitation or professional use. Contact on tel: +3069******* (Viber, WhatsApp), Website: www.megahouse.gr
More

Here are the results we got after running the feature extraction on that listing:

1. Public transport stations: Yes, it is located a 10-minute walk from AMPELOKIPI METRO station.

2. Amenities: Yes, it is near the Evelpidon Courts, Hospitals, and a main road with frequent public transportation.

3. Layout: Yes, it has a living room, kitchen, entry room, 2 rooms (1 can be used as an office), communication corridor, and bathroom.

4. House type: Yes, it is an apartment.

As we can observe, the model successfully identifies the presence of the features and lists them to the user.

Price Prediction Model

Having concluded the training process, it was important to evaluate our model on the holdout dataset containing 10% of the total collected data, to see how it generalizes over unseen data. This process would give us a good idea on the model's performance in a real world scenario. By predicting the price bins of the holdout dataset, we got a final test validation of 51,5%. This indicates that the model predicts correctly the class labels by approximately 51,5% in the unseen data and we understand that for our project this is better than random guessing which is equal to 20% because we have five categories. A better way, however, to visualize the results of the predictions is through a confusion matrix.

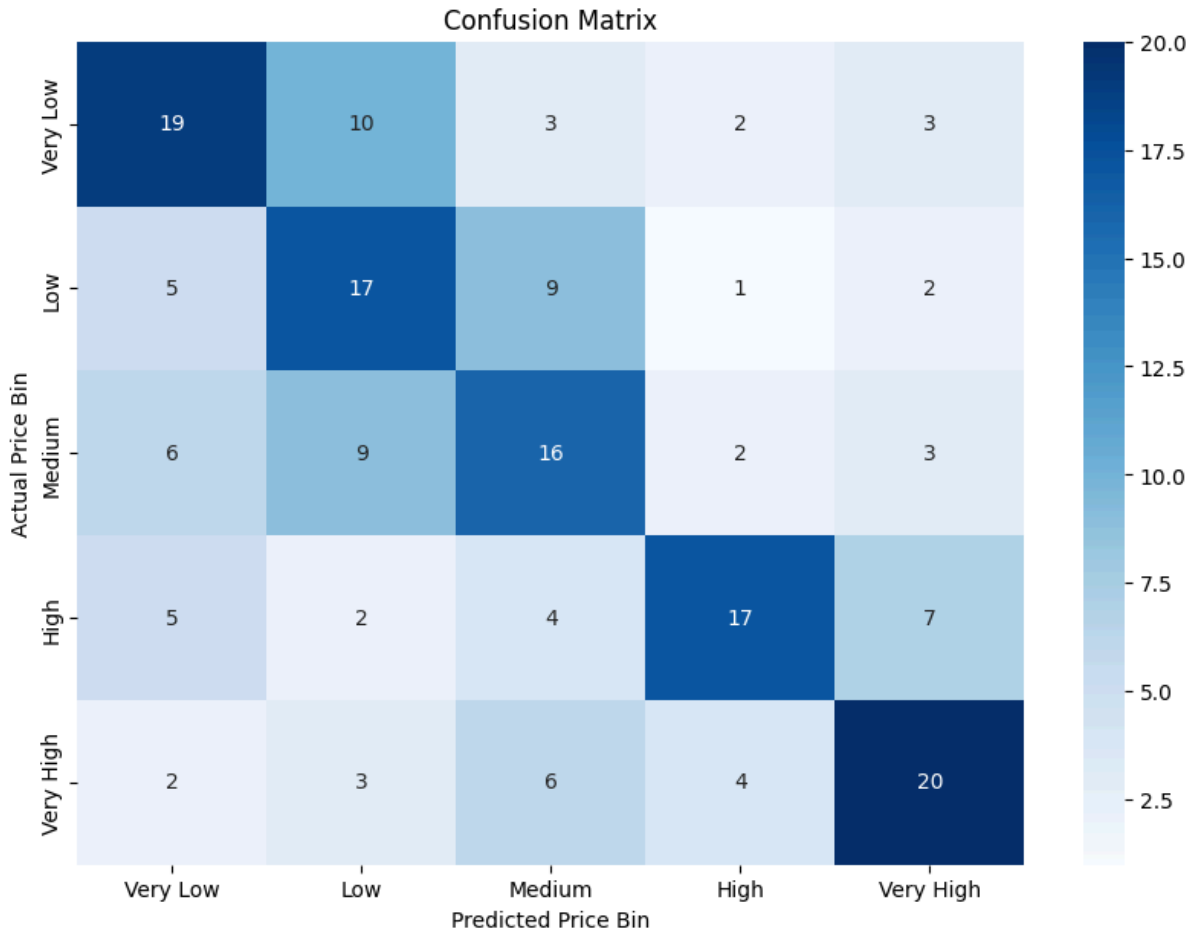


Figure 7: Confusion Matrix of model predictions

The confusion matrix helps us to evaluate the performance of our classification model. Each row represents the instances in an actual class while each column represents the predictions. The diagonal of the matrix therefore represents all instances that are correctly predicted. From our confusion matrix we can understand that in rows we have the true class labels which are the Very Low, Low, Medium, High and Very High whereas in columns we have the predicted class labels with the same categories as in true class labels. So, in the diagonal we have the correct predictions and more specifically for the category Very Low we have 19 correct predictions, for the Low we have 17 correct predictions, for Medium we have 16 correct predictions, for High we have 17 correct predictions and for the Very High we have 20 correct predictions. With this output we realize that our model predicts better in class Very High, because we have a higher number of correct predictions compared to the other classes.
The confusion matrix provides a much clearer view of the model's performance. The primary diagonal is distinct, meaning, in most cases, the model correctly identifies the price category of the property. Additionally, in the majority of cases where there is misclassification, the predicted category of the property is its immediate neighbor. If we take, for example, the Very Low category, which contains properties of up to 75.000 €, out of a total of 37 cases, 19 are correctly labeled, while another 10 are marked as Low. If we take into account that the price separation is quite arbitrary, we conclude that the model's performance is perhaps better than the accuracy would indicate. Two houses priced at 74.000 and 76.000 € respectively, would be in different categories, yet likely exhibit very similar characteristics in terms of images and description.
Another way to evaluate the model's performance would be a metric that penalizes misclassification according to the difference of the predicted label from the actual one. For example, misclassification of a Very Low property as Low should not weigh as much as misclassifying it to Very High. For this reason, we calculated another metric, the Mean Square Error, or MSE for each category, which is calculated from the following formula:

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}\left(Y_i - \hat{Y}_i\right)^2$$

MSE(*Very Low*) = 2,38
MSE(*Low*) = 1
MSE(*Medium*) 1,31
MSE(*High*) = 1,83
MSE(*Very High*) = 2,49

MSE(*Overall*) = 1,8

From the MSE results, it appears that the model is more effective in distinguishing between houses in the middle of the price range. Even though it is more likely to successfully classify a house in the extremes of the price range, as shown by the confusion matrix, misclassification in those cases could be more costly.

5.2. Qualitative & Error Analysis

Price Prediction Model

By reviewing the results of the model's predictions on the holdout dataset, we came to multiple conclusions. The primary one being that the model has moderate success in learning from text and images in order to classify properties according to their price range. In order to get a definitive indication of the model's performance, we would need to benchmark it against human accuracy. However, because of time restrictions, we were unable to conduct proper large scale testing with humans in order to compare the model's results.

Traditionally, more robust models for price prediction have been made using simpler methods, such as linear regression, and numerical features for inputs, such as number of rooms, square feet etc. That said, our model leverages new forms of data, and can potentially be used in an ensemble model to provide better classification results. Another potential usage for this model would be to examine the cases where there is a mismatch between the price assigned by the user and that of the model. In cases where the model undervalues the property, in particular, it could be because the pictures and description are not representative of other listings in its price category. The user could thus be notified that their content could be enhanced in order to improve the ad's quality.

As part of the error analysis conducted, we implemented a function that returns the composite image and the description from a misclassified house, chosen randomly. This function was called many times in order to perhaps infer the reasons behind the incorrect labeling. From what we could tell, properties that presented a variety of rooms in the composite image, meaning having a variety of different pictures in the first 5 which appear on the page, were evaluated more favorably by the model, often giving them a higher price estimation than the actual one. The opposite was also true for listings which had limited variability in the images of the composite. In those instances, the model often undervalued the property. This makes sense, since more expensive houses usually have more rooms, and thus more variety in the pictures presented in the listing. However, if the first images appearing on the user's page do not reflect that, the model's response is more negative. That said, as a user browsing through listings, the first images of a property are crucial in order to formulate a first impression. For that reason, the model could prove useful to potential ad creators to reorder their uploaded images in order to better convey the property's true value from the start.

In the image below, for instance, the property is evaluated at the Low price range (75.000-140.000 €) but it actually is in the Medium price range (140.000-230.000 €). As we can see, there are 3 pictures of the same room, one of which is the product of image augmentation. As a result, the actual listing contains just 3 photos, with 2 of them being of the same room.

On the contrary, the property shown in the image below, receives a Medium classification (140.000-230.000 €) while actually being in the Very Low range (below 75.000 €). We hypothesize that since the images showcase four distinct and different rooms (kitchen, bathroom, living room, entry), the model ranks it higher than its actual price.

6. Discussion, Comments/Notes and Future Work

6.1. Obstacles we Faced

Problems with captcha

When we first ran the scraper to create our dataset, we soon realized that there was a problem that we had to solve first. When we would scrape the data from a listing and move to the next, CAPTCHA would understand that it was not a human accessing the webpage, so it would flag us as bot and request us to manually complete CAPTCHA verification in order for the process to continue. It was not possible for a team member to be able to manually complete those verification requests every time for 600 listings, so we had to come up with a solution.
That solution was to use a few tools that would imitate some human behavior during the scraping process. That means to add random scrolling in the webpage, random clicks throughout the session and also saving the cookies of the current session and loading them onto the next one. That way we were able to overcome the problem and carry on with our tasks.

Scraper is able to check the first five images

The scraper was only able to retrieve at maximum 5 images from each property listing. This happened because each listing page loads with the first 5 images visible and the user has to manually press a button in order to load the rest of the gallery. While still solvable, this required a level of sophistication in the implementation we had not foreseen until we had finished gathering the data and were on our way to train the models. This could potentially lead to problems if those 5 first pictures were not representative of the entire property. For example, if the first pics were only about the bathroom and the kitchen of the house, while the other could be pictures about a garden, balcony or other rooms, then that would lead to inaccuracies in our models, since the model would get information only about specific parts of the listing and omit important information.
While we were not able to solve that problem head on, we decided that in order for a listing to be of high quality, the first images appearing on the page should be representative of the property.

Non available listings

During the creation of the dataset, our team had to manually collect links of house listings in an EXCEL file. The first goal was to stop at around 700 listings and then feed them to the scraper in order to create the main dataset and begin with the analysis of it.
Unfortunately, as the process of collecting the links of the ads took a few days to complete, some of the houses were sold, so after running them through the scraper, the data were not available to be used in the dataset and as a result, our dataset fell short by about 100 listings from our first goal.

Hallucination problems

Here, during the paraphrasing step, the non-fine tuned LLaMA model, quite often would generate information in the description that was not requested, or that would not have any connection with the context. Occasionally, we noted that it would add extra information to the amenities that were not present in the original description of the house listing.
That problem happens when the model tries to fill in gaps based on patterns it learned from general training data, rather than following patterns and info that are in specific domain input data.
We solved that problem by following a fine tuning technique that we have presented above in chapter 3.1.(Feature extraction model fine tuning methodology).

Price Prediction Accuracy

While the price prediction model can successfully extract image and text features in order to give a price classification, its accuracy may be further increased. It seems that with the available data and architecture we reached a hard ceiling at around 51% test accuracy. In a future scenario where we had more available training data, as well as resources to develop and fit more complex models, we could further optimize it to increase the predictive performance.

Image Representation

While developing the CNN branch of the price prediction model, we considered the input shape a lot. One consideration was, instead of feeding the images themselves, to have as input a low dimension representation of them. For this purpose, we explored the idea of using a Variational Autoencoder (VAE) in order to transform the images into a latent space representation. However, after some trials, we found out that training our own VAE was very resource consuming and we could not find a pre-trained one that fit our scenario reasonably well. As part of future development, we would like to explore this idea further, provided we had the computational resources necessary.

6.2. Future Work

Looking ahead, our main goal would be to use this project as a proof of concept to collaborate with real estate platforms like Spitogatos, or some other similar websites, where our models could be applied. That type of collaboration could help us with improving price prediction accuracy, could increase the quality and quantity of the dataset that we are working with and could lead to even better and complete descriptions. Better access to data and resources would resolve a lot of the limitations we faced during development and undoubtedly lead to better models and overall results. Additionally, we aim to expand our

dataset to include more diverse listings from all neighborhoods around or even outside Athens, in order to make our model even more robust.

7. Project Management

7.1. Members/Roles

Konstantinos Toulis:
Web scraper implementation, composite images and image augmentation, Price Prediction model implementation

Viktor Kalatzis:
Exploratory Data Analysis and report of image classification, composite images and Price Prediction in sections of dataset overview, methodology and results

Ioannis Tsougkrianis:
Description augmentation for synthetic data based on real data, LLM fine-tuning, all LLM functions and implementations (image description, user and image description comparison, user and image description combination)

Athanasios Kostarelis:
Reporting about sections on methodology, experiments, results, and discussion. Managed the overall structure and flow of the document, ensuring clarity and cohesion. Coordinated with team members to incorporate technical details, summarizing their contributions into a comprehensive report.

7.2. Time Plan

Data Collection (Weeks 1-4)
Week 1: Gathering of different property listing URLs from spitogatos (around 700 ads).
Week 2-3: Web scraper implementation and dataset creation from list of URLs (around 600 ads).
Week 4: EDA

Data Preprocessing (Weeks 5-6)
Week 5: Image standardization, augmentation and composite image creation.
Week 6: Descriptions augmentation through Groq API and Llama 3.

Models Implementation (Weeks 7-11)
Week 7: LLM fine-tuning for description feature extraction
Week 8: Implementation of house image features and description comparison
Week 9-11: Implementation of price prediction model

Report (Weeks 12-13)

8. Bibliography

Groq model:

- https://huggingface.co/Groq/Llama-3-Groq-70B-Tool-Use

Fine-tuning:

- https://huggingface.co/blog/mlabonne/sft-llama3
- https://huggingface.co/jtsoug/llama3.1_8b_real_estate_feature_ft

all-MiniLM-L6-v2:

- https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2

Keras:

- Developer guides (keras.io)

Price estimation from pictures:

- https://www.kaggle.com/code/amir22010/house-price-estimation-from-image-and-text-feature

Keras: Multiple Inputs and Mixed Data:

- Keras: Multiple Inputs and Mixed Data - PyImageSearch

LLM fine-tune:

- https://www.ml6.eu/blogpost/fine-tuning-large-language-models
-]https://ubiops.com/when-should-you-fine-tune-your-llm/

llava-v1.5-7b:

- https://huggingface.co/liuhaotian/llava-v1.5-7b

Hosting LLM on Google Colab:

- https://betterprogramming.pub/set-up-an-llm-project-using-a-free-gpu-in-google-colab-e55453bfc760
- https://shiv248.medium.com/hosting-open-source-llm-model-on-google-colaboratory-cc14a42d4053