



# Price Discovery Product



**Notion Tip:** Use this template to plan and execute every part of your launch with your team, in one, centralized page.

## Table of contents

- [1. Introduction](#)
  - [1.1 Background](#)
  - [1.2 Objective](#)
  - [1.3 Team](#)
- [2. Approach](#)
  - [2.1 Scope](#)
- [3. Data](#)
  - [3.1 Data Collection](#)
  - [3.2 Vector Databases](#)
  - [3.3 Knowledge Graph Database](#)
- [4. Architecture](#)
  - [4.1 CLIP + Statistical Model](#)
    - [4.1.1 Workflow Overview](#)
  - [4.2 Retrieval-Augmented Generation](#)
    - [4.2.1 Components of RAG](#)
    - [4.2.2 Steps Involved In RAG](#)
    - [4.2.3 Workflow Overview](#)
  - [4.3 Agents](#)
    - [4.3.1 Workflow Overview](#)
    - [4.3.2 Agent Components](#)
- [5. Training Process](#)
- [6. Deployment](#)
  - [6.1 Using Docker Compose](#)
    - [6.1.1 Docker Compose Configuration](#)
  - [6.2 Makefile Commands](#)
  - [6.3 Monitoring](#)
- [7. Experimental Results](#)
  - [7.1 Evaluation Metrics](#)
    - [7.1.1 Mean Absolute Percentage Error \(MAPE\)](#)
    - [7.1.2 Reasoning score](#)
    - [7.1.3 BLEU and ROUGE score](#)
    - [7.1.4 Context Relevance score](#)
  - [7.2 Performance](#)
- [8. Future Work](#)
  - [8.1 Short Term](#)

8.1.1 Experiment with Different LLMs
8.1.2 Vector Embeddings
8.1.3 Vector Databases
8.1.4 Prompt Engineering
8.2 Medium Term
8.2.1 Knowledge Graph Database
8.2.2 Agents Experimentation
8.3 Long Term
8.3.1 Fine Tuning the LLM
8.3.2 Latency on the Scraper
9. References

# 1. Introduction

## 1.1 Background

Mastering the art of pricing is essential for any business to thrive. Getting the pricing right can boost sales and set the stage for long-term success. Different companies employ various pricing strategies tailored to their specific needs and markets. Typically, these strategies rely on tried-and-tested formulas adaptable across different product lines and market segments. However, there's always the danger of pricing products too low or too high, which can harm a company's reputation and bottom line. Strike the right balance, and your business will flourish.

## 1.2 Objective

In this thesis, we've developed a price prediction model aimed at assisting companies in determining the ideal pricing for their latest products. By analyzing product design images and description, our model helps companies ensure their products are competitively priced in comparison to similar offerings from competitors. We train our model by inputting description and images from existing products, allowing it to accurately predict prices for new products based on prevailing market standards. It's like having a crystal ball for pricing, but powered by machine learning.

## 1.3 Team

The members of the Price Discovery team were Thamanna Hafeez, Harmanan Kohli, Esther Wabomba, O'Neil Mbakwe (project coordinator), Kimani Kibuthu, Liguang Li, Yunmo Koo, Genyuan Liu, Trung Nguyen, Victor Onuorah, Aditya Shingote, Sudip Pokhrel, Treyton Wofford, Kwan Lee.

# 2. Approach

## 2.1 Scope

The scope of the project is to create an LLM that reads a product's image and description to generate a price range for that product. The price range should have the following features.

- Range should be given in USD currency

- LLM should take up to one image
- The description should be in the English language

## 3. Data

### 3.1 Data Collection

In this section, we delve into the unique characteristics of the dataset crucial for addressing our research inquiry. While scouring through numerous public datasets on fashion products, we encountered a common issue: many lacked the integration of images and metadata. Additionally, numerous datasets had empty pricing columns. Although there were plenty of product review datasets, they often lacked the detailed features and characteristics essential for our analysis. To overcome these challenges, we carefully selected and curated the data utilized in this thesis project from a variety of sources. The dataset that we ended up selecting was **Amazon Products Dataset 2023**. This dataset contained 1.4M products which included product image URL, price, and title (which we used as description). A major concern with the Stylish Product Image dataset was that the currency wasn't US Dollars and hence went with the Amazon Product dataset 2023. Stylish Product Image Dataset was used earlier for testing the Clip + Statistical Architecture and its result can be found in the Architecture section for Clip + Statistical Architecture.

#### 3.1.1 Scraper for Data Collection

For downloading the images from the Amazon website, a scraper was created using Selenium in Python. The URL for the images is already present in the CSV file provided on the Kaggle.

The link for the Amazon Product Dataset 2023 and Stylish Product Image Dataset can be found in the references section.

The requests library was also used for getting the images and Product description and Product Details but were unable to do so as it was returning a message from the Amazon website to use the API. So Selenium was used to automate the task of collecting more information.

The code is available in the `amazon_product_scraper.ipynb` under the notebooks folder in `data_preparation_notebooks` in GitHub.

#### 3.1.2 Recommendations

One point to note is that as Selenium opens up a browser session, it is recommended to use small start and end values. If going for bigger values, it might lead to errors, so it is recommended to save intermediate results.

## 3.2 Vector Databases

Vector database includes 50,000 products with its title and image. The embedding function used to get the vectors for text and image is OpenAI's OpenCLIP embedding.

#### 3.2.1 Title/Text Vector Database Collection

The text vector database collection is composed of price, ASIN(Amazon Standard Identification Number), and title.

### 3.2.2 Image Vector Database Collection

The image vector database is composed of price, ASIN(Amazon Standard Identification Number), image URL, and title.

### 3.2.3 Implementation

Vector Database was implemented into the AI Agent. The description of a product is given to the agent who initializes the search on the database first, then moves to other tools like Google search.

Its implementation can be found in the `agents_notebooks` branch within `fellowship/price-discovery/notebooks` GitHub repository. There is an `Agents.ipynb` notebook with code implementations.

Vector Database was also implemented into the LLM of the RAG architectural model. The product description was given to the LLM in a similar way as to the agents, whereby the LLM searches for similar products in the database and if not found, then moves to Google Search. The implementation is found in the `RAG-Complete-Open-clip-vit-b-32-chroma.ipynb` notebook within the `rag_notebooks` branch

### 3.2.4 Results

When the agent's performance is compared with and without the database search, the price range of the product with the database search is narrower than without. Both with and without gave price ranges that included the true price.

### 3.2.5 Future Considerations

The current vector database in Qdrant is straightforward forward with most of the product title texts uncleaned. Cleaning up the product title and extracting information can be a very possible improvement to the current database. Additionally, experimenting with different embedding functions would be a good consideration as better functions will be developed in the future.

## 3.3 Knowledge Graph Database

In this section we will describe the knowledge graph database. Knowledge graphs represent information in a structured format, typically using nodes (entities) and edges (relationships) to illustrate how different pieces of information are connected. It's adept at integrating information from various sources, even if the data is structured differently, enabling a unified view of information. In this case we used Neo4j as our knowledge graph.

### 3.3.1 Experimentation

We used a data scraper to extract products from Amazon in 10 different categories. Since having the data we have tried different schemas, Neo4j allows for a flexible schema, making it easier to evolve our data model over time. This flexibility is particularly beneficial in scenarios where the structure of our data is complex or changing, enabling us to add new types of relationships and entities without significant rework.

### 3.3.2 Results

By representing products, categories, brands, attributes, and price ranges as nodes in a graph, Neo4j enables us to explore the connections between these entities in ways that traditional databases cannot. This can uncover insights such as which attributes are most common for a certain brand, how products are distributed across different price ranges, or how changing an attribute (like color or size) affects the product's placement within a price range.

### 3.3.3 Limitations

- Scrapped data from Amazon's product descriptions often lacked crucial details like brand name, weight, and color, leading to inconsistencies and missing information.
- Using Selenium for automation proved inefficient for scraping a wide range of products simultaneously, making the data collection process more cumbersome.
- The lack of sufficient, consistent data and irregular values hindered the effectiveness of utilizing a Knowledge Graph (KG) architecture, which relies on abundant attributes and detailed product data for fast and efficient information retrieval. This limitation prompts reconsideration of KG's suitability for future improvements.

### 3.3.4 Considerations

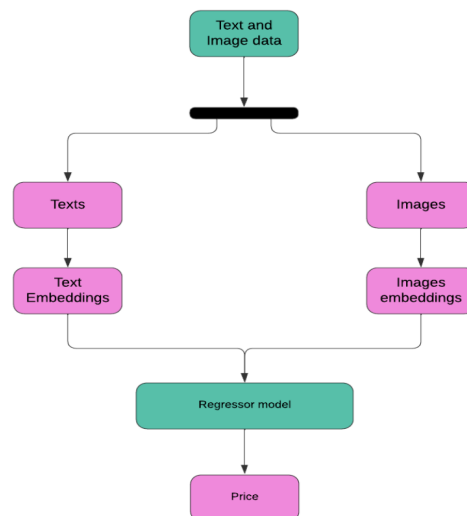
- A method to try is to have the AI agent build the knowledge graph over time by query. This makes it so that it only takes in the important factors of the product and the info in the knowledge graph stays up to date. This will require trying different prompt engineering techniques for this process.

## 4. Architecture

### 4.1 CLIP + Statistical Model

CLIP is a pre-trained model for telling you how well a given image and a given text caption fit together introduced by OpenAI. It was **trained contrastively** on about 400 million web-scraped data of image-caption pairs. There are four different CLIP models: a ViT-B/32, a ViT-B/16, a ViT-L/14, and a ViT-L/14@336px. We used the ViT-B/32, and the ViT-L/14 in this project, corresponding code can be found at [notebooks/clip\\_notebooks/price\\_discovery\\_CLIP-vit\\_large-stylish\\_products.ipynb](#), and [notebooks/clip\\_notebooks/Price\\_Discovery\\_CLIP-vit-base-patch32.ipynb](#)

## CLIP + Statistical Model



### 4.1.1 Workflow Overview

1. **Input Acquisition:** Amazon Product Dataset 2023 and Stylish Product Image Dataset
2. **Image and text Analysis with Visual VLLM:** The product image and description undergo comprehensive analysis by the CLIP models to extract detailed features and characteristics and convert the image and text information into vector embeddings.
3. **Combination with Description:** Extracted image embeddings are combined with text embeddings, creating more comprehensive and enriched product embeddings.
4. **Price Range Determination:** Based on product embeddings, statistical models (random forest model, XGboost model, and LightGBM model) are trained to get the price and price range (5 - 95% or your customized percentile range).
5. **Result Delivery:** The combination of the LightGBM model and ViT-L/14 model had the best performance in predicting the price, with an R square of 0.74 and mean\_absolute\_percentage\_error of 0.27. However, it did a bad job of predicting the price range of a product.

## 4.2 Retrieval-Augmented Generation

LLMs, or Large Language Models, are a crucial artificial intelligence technology that drives intelligent chatbots and other natural language processing (NLP) applications. However, LMs can provide unpredictable responses due to their nature. Furthermore, the training data for LMs is static and has a cut-off date, limiting the knowledge they can acquire. RAG involves enhancing the output of a large language model by consulting an authoritative knowledge base outside of its training data to produce a response i.e., RAG helps these models by allowing them to search for additional information from the Internet or other places, making their answers more accurate.

### 4.2.1 Components of RAG

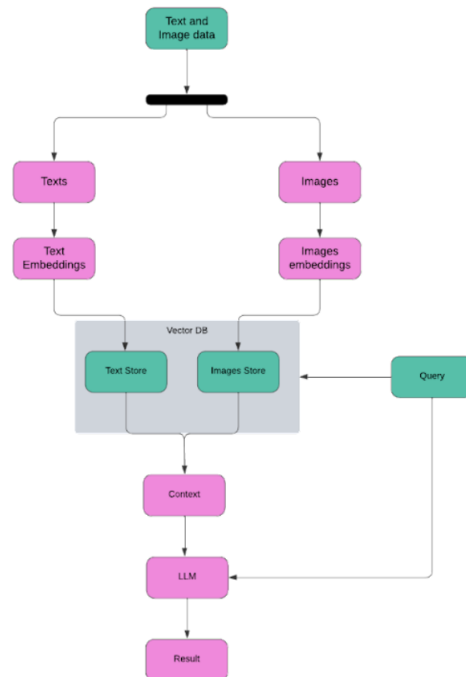
- **Retriever:** The retriever module receives the query and retrieves what it evaluates to be the most accurate information based on a store of semantic search vectors from the vector database.
- **Generator:** The generator combines the received embeddings with the original query and processes them through the trained language model to produce a more informative output.

#### 4.2.2 Steps Involved In RAG

- **Create external data:**
  - The new data that is not part of the LLM's original training data set is referred to as external data. The external data we have exists in the form of a vector database on Qdrant which is encoded from the Amazon Product Dataset 2023. The embedding language models convert data into numerical representations and store it in the vector database. This process creates a knowledge library that the generative AI models can understand. Currently, we have 50,000 records of text and image embeddings on the vector database.
- **Retrieve relevant information**
  - The next step involves performing a semantic search by converting the user query into a vector representation and matching it with the vector databases. The text/image query was encoded using the same embedding function used during the database creation.
- **Augment the LLM prompt**
  - The RAG model enhances user input by incorporating relevant retrieved data to provide context. This step requires prompt engineering techniques for effective communication with the LLM. By augmenting the prompt, the large language model was able to generate accurate price ranges in response to the user queries.

#### 4.2.3 Workflow Overview

# RAG



- Input Acquisition: An input query was made in the form of a text description and image URL
- The query is passed to the LLM (model=" Vit-B-32", pretrained = 'openai') from Open-CLIP.
- The RAG model encodes the query into text and image embeddings, which are then passed into the vector database using any one of the following two ways:
  - First, only the text query is encoded and compared with the vector database.
  - Second, a new enriched description is generated from the queried text and image analysis using a VLLM for a more precise result. In this case, Gemini pro-vision was used as the VLLM.
- The retriever decides the most relevant product using semantic search abilities within the database. If a similar product is not found, it searches the internet and gets the information. It then sends the parsed embeddings to the generator.
- The generator combines the embeddings with the original query to provide context and then delegates its task to the language model to generate the most relevant output for the user. The LLMs we experimented with here are Microsoft Phi2, Gemini and Mixtral.
- Result Analysis: The model produced fairly good results with most products, but with certain other products not in the database, results were a little amiss. Furthermore, inconsistencies in the price range were seen for the same product. Results were based on the preciseness of the input description, a generalized description gave a generalized price range whereas an accurate description resulted in an accurate price range. More complex prompt engineering techniques and ways to acquire unambiguous text descriptions are required for decent results.



## 4.3 Agents

The Price Discovery project incorporates a React agent designed to facilitate efficient price range determination for products based on user inputs. Instantiated using LangChain, the agent leverages advanced tools and an LLM (such as Mixtral or Gemini) to analyze product descriptions, images, and search results. It reasons about the collected data to determine a price range while giving reasoning behind its answer.

### 4.3.1 Workflow Overview

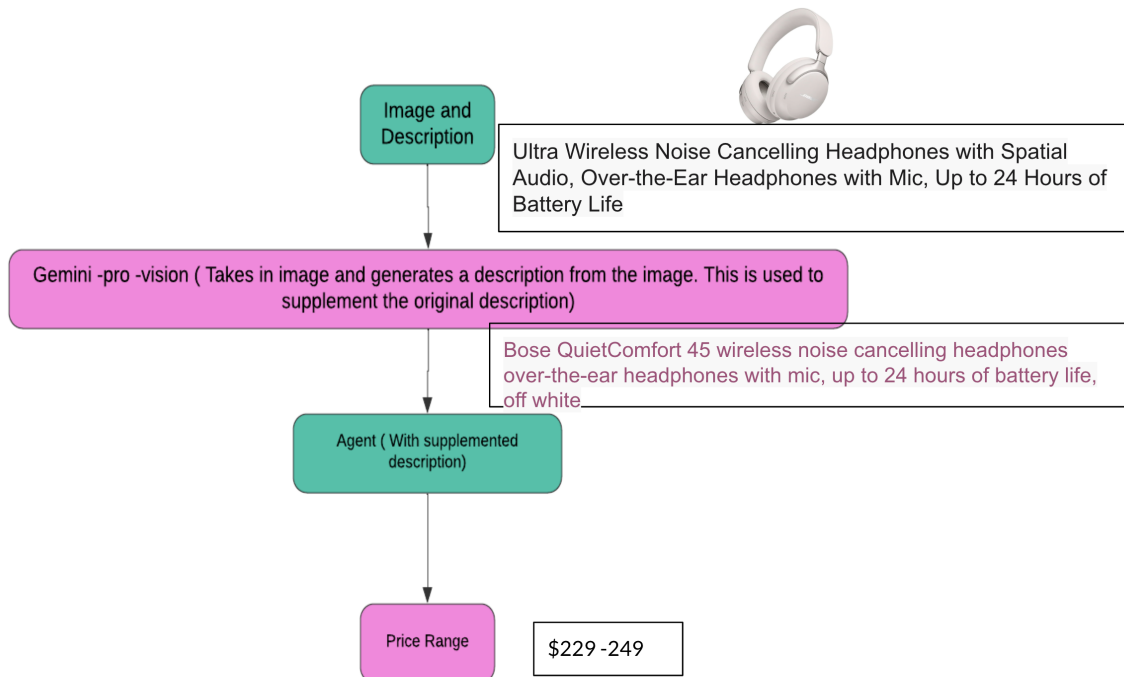
1. **Input Acquisition:** Users input a product image and its description via the Streamlit app interface.
2. **Pipeline Initiation:** Upon submission, the system triggers a POST request to the FastAPI app, initiating the pipeline process.
3. **Image Analysis with Visual VLLM:** The product image undergoes comprehensive analysis by the Visual Large Language Model (VLLM) in this case Gemini pro vision to extract detailed features and characteristics.
4. **Combination with Description:** Extracted image details are combined with the provided product description, creating a more comprehensive and enriched product overview.
5. **Agent Operations:**
  - **Internet Search:** The agent conducts searches on the internet to gather information about similar products available online.
  - **Image Database Search:** Leveraging image databases, the agent seeks out visually similar products for comparison.
  - **Text Database Search:** Text databases are queried to identify products with similar descriptions or features.
6. **Price Range Determination:** Based on the findings from the internet search, image database search, and text database search, the agent determines a price range for the product under consideration.
7. **Result Delivery:** Once the price range determination process is completed, the agent returns the calculated price range along with the rationale behind it to the user via the Streamlit app interface.

### 4.3.2 Agent Components

The agent is comprised of the following key components:

- **Internet Search Tool:** Responsible for conducting searches on the internet to gather information about similar products.
- **Image Database Search Tool:** Utilizes image databases to find visually similar products for comparison.
- **Text Database Search Tool:** Queries text databases to identify products with similar descriptions or features
- **LLM:** In this case, mixtral or Gemini is used.

- **Prompt:** A prompt gives the agent specific instructions on how to work. The prompt is a ReAct prompt.



## 5. Training Process

No training was undertaken as basic prompting was enough.

The prompt used for the vision LLM was as follows:

```

vision_chat_template = HumanMessage(
    content=[
        {
            "type": "text",
            "text": f""Your task is to generate a searchable prompt about the
1. Given the description {description}, extract relevant details about the
2. Use these details and the given description to generate a concise and v
3. Return ONLY the searchable prompt in the following JSON format: {"sear
Please strictly follow these instructions and provide your response in the
        },
        {"type": "image_url", "image_url": image}
    ])
  
```

The prompt used for the agent was as follows:

```
test_prompt_seven = """
As an experienced product analyst proficient in determining accurate products, please provide the following information:

{{
  "price_range": "<price range as a number range>",
  "reason": "<reason for the price range>"
}}

To generate this output, follow these steps:

1. Utilize all available tools {tools}, including internet searches and data analysis.
2. Identify the most similar products and use them as a basis to come up with a price range.
3. After analyzing comparable products, generate a highly accurate, reasonable price range.
4. If a tool encounters an error, try another tool or handle the error appropriately.
5. Replace <price range as a number range> with your estimated price range.
6. For <reason for the price range>, Please provide a comprehensive rationale for the price range.

Your output should strictly follow the specified JSON format and include only the JSON object.

Use the following format:

Question: the input question you must answer.
Thought: you should always think about what to do.
Action: the action to take, should be ONE of or ALL of [{tool_names}].
Action Input: the input to the action. It SHOULD be a string.
Action: choose another tool if a tool fails or handle its error.
Observation: the result of the action, including insights gained or findings.
... (this Thought/Action/Action Input/Observation can repeat N times)
If no result is found, use your own knowledge.
Thought: I now know the final answer that is based on the in-depth analysis.
Final Answer: the final answer to the original input question in the specified format.

Begin!

Question: {input}
Thought: {agent_scratchpad}
"""
```

## 6. Deployment

### 6.1 Using Docker Compose

Docker Compose is a tool for defining and running multi-container Docker applications. It simplifies the process of managing complex application stacks by allowing you to define the services, networks, and volumes required for your application in a single YAML file.

### 6.1.1 Docker Compose Configuration

Create a `docker-compose.yml` file in the root directory of your project with the following content. Where the repo name is your repo on docker.

```
version: '3.8'

services:
  fastapi:
    image: repo/price-discovery-fastapi:latest
    build:
      context: .
      dockerfile: Dockerfile.fastapi
    ports:
      - "8000:8000"
    env_file:
      - .env

  streamlit:
    image: repo/price-discovery-streamlit:latest
    build:
      context: .
      dockerfile: Dockerfile.streamlit
    ports:
      - "8501:8501"
    env_file:
      - .env
```

This configuration defines two services: `fastapi` and `streamlit`, each with its own Dockerfile, ports, and environment variables.

The Dockerfiles are as follows:

1. FastAPI app (Dockerfile.fastapi):

```
# Use the official Python image as the base image
FROM python:3.9-slim

# Set the working directory in the container
WORKDIR /app

# Copy the poetry files to the working directory
```

```

COPY poetry.lock pyproject.toml ./

# Install Poetry
RUN pip install poetry

# Install the project dependencies
RUN poetry config virtualenvs.create false && poetry install --no-interact
RUN pip install fastapi

# Copy the FastAPI application code to the working directory
COPY fastapi_app.py .
COPY src src/
COPY configs configs/
COPY logs logs/

# Expose the port for FastAPI
EXPOSE 8000

# Set the entrypoint to run FastAPI
CMD ["poetry", "run", "uvicorn", "fastapi_app:app", "--host", "0.0.0.0", ""]

```

## 2. Streamlit app (Dockerfile.streamlit):

```

# Use the official Python image as the base image
FROM python:3.9-slim

# Set the working directory in the container
WORKDIR /app

# Copy the poetry files to the working directory
COPY poetry.lock pyproject.toml ./

# Install Poetry
RUN pip install poetry

# Install the project dependencies
RUN poetry config virtualenvs.create false && poetry install --no-interact
RUN pip install streamlit

# Copy the Streamlit application code to the working directory
COPY streamlit_app.py .
COPY src src/
COPY configs configs/
COPY logs logs/

```

```
# Expose the port for Streamlit
EXPOSE 8501

# Set the entrypoint to run Streamlit
CMD ["poetry", "run", "streamlit", "run", "streamlit_app.py", "--server.port=8501"]
```

## 6.2 Makefile Commands

To streamline Docker operations, you can use a Makefile with the following commands:

```
# Build Docker images using docker-compose
build:
    docker-compose build

# Push Docker images to the registry using docker-compose
push:
    docker-compose push

# Pull Docker images from the registry using docker-compose
pull:
    docker-compose pull

# Run the Docker containers using docker-compose
run:
    docker-compose up -d

# Stop and remove the Docker containers using docker-compose
stop:
    docker-compose down

# Remove the Docker images using docker-compose
clean:
    docker-compose down --rmi all
```

## 6.3 Monitoring

You can monitor the performance on [langsmith](#).

# 7. Experimental Results

## 7.1 Evaluation Metrics

The set of evaluation metrics can be divided into two groups

- **To evaluate the final answer:** Price Range + Reason

- Mean Absolute Percentage Error: for the Price Range
- Reasoning score (LLM-based): for the Reason
- **To evaluate the contexts:** How relevant the retrieved product descriptions from the databases or the internet are to the original product description
  - BLEU and ROUGE score
  - Context Relevance score (LLM-based)

*Note:*

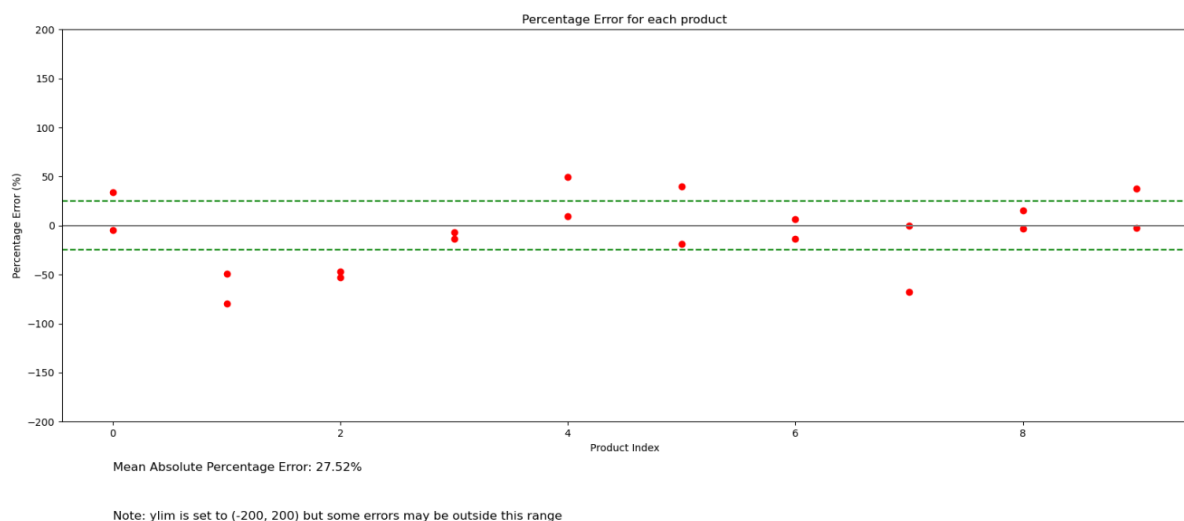
- Details of the implementation of each metric can be found in the notebook `evaluation_metrics.ipynb`
- Both LLM-based metrics are adapted from a LLM-evaluation library called [Trulens](#) (Details in the notebook)

### 7.1.1 Mean Absolute Percentage Error (MAPE)

Calculate the percentage error of the min and max of the predicted price range of each test product, then take the average of them

```
# Psuedo-code
percentage_errors = []
for product in products:
    percentage_errors += [error(min_price), error(max_price)]
mape = sum(percentage_errors) / len(percentage_errors)
```

We can also visualize the percentage errors using a plot



Each two-dot represents the min and max of a predicted price range for a particular products (in this case, there are 10 products)

The closer the red dots are to the middle zero-line, the better the performance. If the zero line is between the two dots, then the test price is inside the range

Two green dashed lines are drawn at the 25% error level for reference.

One drawback of MAPE is that it is **susceptible to outliers**. If the original price is \$10 and the predicted price is \$100, then the percentage error is 900%. But if the original price is \$100 and the predicted price is \$10, then the percentage error is only 90%

### 7.1.2 Reasoning score

Prompt the LLM to provide a score (0 to 10) to evaluate the quality of the reason generated by the model for the predicted price range.

Criteria include: Clarity, Justification, Completeness, Accuracy, Level of Detail, Step Validity, Step Completeness

### 7.1.3 BLEU and ROUGE score

These scores measure how similar two product titles are based on word match (not semantic meaning)

BLEU score: Calculate the average between the precision at 1-gram, 2-gram, 3-gram, and 4-gram

ROUGE score: Calculate the average between ROUGE-1, ROUGE-2, and ROUGE-L

### 7.1.4 Context Relevance score

Prompt the LLM to provide a score (0 to 10) to evaluate how relevant the retrieved product is to the original product

Criteria include: Product Type, Functionality, Target Users, Style, Material, Brand

## 7.2 Performance

Due to our limitation in time and resources, we haven't been able to perform a formal evaluation on a large amount of data. Specifically, since SERP API only gives 100 free search a month, evaluating agents with internet search has been difficult.

From our experience, the quality of the price range usually depends on a few factors:

- The input products: Products with stable prices, such as t-shirts, typically perform better. In contrast, products with highly variable prices, like watches, often perform worse.
- The retrieved products from the databases or the internet: If the agent is able to retrieve similar products to the input product, the performance will be better
- The reasoning ability of the LLM: However, this factor may not be as important as the above two. We have tried two LLMs: Mixtral and Gemini with multiple different prompts but the reasoning score or the MAPE of each architecture is about the same



For reference, here are some approximate of the average score ranges for different metrics based on our experience

- MAPE: ~0.4 - 0.6
- Reasoning score: ~0.8 - 0.9
- BLEU score: ~0.15 - 0.25
- ROUGE score: ~0.35 - 0.45
- Context relevance score: ~0.75 - 0.85

Here is an evaluation example of RAG using Mixtral or Gemini on a dataset containing 250 products.

Model	ROUGE	BLEU	Context Relevance	Reasoning	MAPE
RAG Mixtral	0.42	0.23	0.78	0.83	0.56
RAG Gemini	0.42	0.23	0.78	0.80	0.48

*Note: The scores of ROUGE, BLEU, and Context Relevance are the same because the context for both architectures is the same in this case*

## 8. Future Work

### 8.1 Short Term

#### 8.1.1 Experiment with Different LLMs

Currently, we used only Gemini and Mixtral in the agents created. When coming to RAG, we worked with Microsoft Phi-2. Gemini, and Mixtral. Moving forward we would work with more LLMs to enhance performance while looking for agents

#### 8.1.2 Vector Embeddings

Currently, we are only able to work with a few embeddings. For text embeddings we worked with all-mpnet-base-v2 and for images we worked with model\_name= 'ViT-B-32', pretrained= 'laion2b\_s34b\_b79k' and pretrained = 'openai' . After getting better results, we switched the text embeddings to use what was used for our images. For future work, we suggest trying ViT-Large embeddings or other image embeddings.

#### 8.1.3 Vector Databases

Initially, we experimented with ChromaDB as our vector database but found that it did not have cloud availability, causing us to move to QDrant. In the future, we suggest trying out different vector databases in order to reduce latency and increase performance.

One more improvement area that can be explored is to add more data in the Vector DB. During our experimentation, the max data that was loaded was 100000 records in both the Text and

Image in Chroma DB. This also was possible because the Vector DB was used locally on a personal laptop. In the case of cloud-hosted Vector Db like QDrant it becomes a challenging task.

### **8.1.4 Prompt Engineering**

For this project, the ReAct type of prompting was utilized. Effective prompt engineering can greatly improve the performance and behaviour of language models on a wide range of tasks.

One key aspect is understanding the model's capabilities and limitations to tailor prompts that play to its strengths while avoiding failure modes. To see if performance improves, it's crucial to experiment with various prompting techniques for the agent and the vision language model. The goal is to find the sweet spot of an optimized prompt with minimal tokens but maximum accuracy. This may involve iteratively refining prompts based on the model's outputs, and adding clarifications, constraints, or corrections as needed. Advanced techniques can further enhance reasoning abilities. Ultimately, effective prompt engineering requires creativity and experimentation to develop prompts that unlock a language model's full potential.

## **8.2 Medium Term**

### **8.2.1 Knowledge Graph Database**

We found that knowledge graph can be enhanced by additional structured and unstructured data sources relevant to Amazon products, such as manufacturer details, brand name, dimensions, color, weight, country of origin, material used, customer reviews, etc. Different graph schemas can be tried with to improve the representation of the attributes.

### **8.2.2 Agents Experimentation**

For agents, we suggest trying out other variants of agents such as plan and execute agents. They can try to play around with different prompts and LLMs. We also suggest trying Corrective RAG.

## **8.3 Long Term**

### **8.3.1 Fine Tuning the LLM**

Currently, we do not do any kind of fine-tuning. For future work, we would recommend fine-tuning LLMs to better fit the tasks and produce better results.

### **8.3.2 Latency on the Scraper**


It was found that as the scraper opened up another session in the Chrome browser, it was having high latency like collecting nearly 250 records in a 1-hour window, so a future improvement or enhancement can be looked is to reduce the time for fetching the required sections. Other libraries can be looked upon like Scrapy.

## **9. References**

*Amazon Products Dataset 2023 (1.4M products)*. (2024, February 17). Kaggle.  
[https://www.kaggle.com/datasets/asaniczka/amazon-products-dataset-2023-1-4m-products?select=amazon\\_products.csv](https://www.kaggle.com/datasets/asaniczka/amazon-products-dataset-2023-1-4m-products?select=amazon_products.csv)

Bratanic, T. (2024, January 19). Enhancing Interaction between Language Models and Graph Databases via a Semantic Layer.  
*Medium*. <https://towardsdatascience.com/enhancing-interaction-between-language-models-and-graph-databases-via-a-semantic-layer-0a78ad3eba49>

Chen, S., Chou, E., Yang, R., & Stanford University. (2021). The Price is Right: Predicting Prices with Product Images. In  
*CS229 and CS221 [Journal-article]*. <https://cs229.stanford.edu/proj2017/final-reports/5237321.pdf>

*Chroma* |  *Langchain*. (n.d.).  
<https://python.langchain.com/docs/integrations/vectorstores/chroma>

*CLIP: Connecting text and images*. (n.d.). <https://openai.com/research/clip>

Demush, R. (2019, February 26).  
*A brief history of computer vision (and convolutional neural networks)*. HackerNoon.  
<https://hackernoon.com/a-brief-history-of-computer-vision-and-convolutional-neural-networks-8fe8aacc79f3>


*Hugging Face – The AI community building the future*. (n.d.). <https://huggingface.co/>

Kehrer, K. (2024, February 29).  
*Best Vector DBs for Retrieval-Augmented Generation (RAG)*. Aporia.  
<https://www.aporia.com/learn/best-vector-dbs-for-retrieval-augmented-generation-rag/>

*LangChain Neo4J Integration - Neo4J Labs*. (n.d.). Neo4j Graph Data Platform.  
<https://neo4j.com/labs/genai-ecosystem/langchain/>

mlfoundations. (n.d.).  
*GitHub - mlfoundations/open\_clip: An open source implementation of CLIP*. GitHub.  
[https://github.com/mlfoundations/open\\_clip](https://github.com/mlfoundations/open_clip)


*Neo4j documentation - Neo4j Documentation.* (n.d.). Neo4j Graph Data Platform.  
<https://neo4j.com/docs/>

*neo4j-semantic-layer* |  *Langchain.* (n.d.).  
<https://python.langchain.com/docs/templates/neo4j-semantic-layer>

Nutan. (2023, September 20). Deep Convolutional Networks VGG16 for image recognition in KERAS.  
*Medium.* <https://medium.com/@nutanbhogendrasharma/deep-convolutional-networks-vgg16-for-image-recognition-in-keras-a4beb59f80a7>

*OpenClip* |  *Langchain.* (n.d.).  
[https://python.langchain.com/docs/integrations/text\\_embedding/open\\_clip](https://python.langchain.com/docs/integrations/text_embedding/open_clip)

Prompt Circle AI. (2024, February 9).  
*Introduction to LangGraph | Building an AI generated*  
*podcast* [Video]. YouTube. <https://www.youtube.com/watch?v=Al6Dkhuw3z0>

*QDrant* |  *Langchain.* (n.d.-a).  
<https://python.langchain.com/docs/integrations/vectorstores/qdrant>

Rustamy, F., PhD. (2024, March 18). CLIP model and the importance of multimodal embeddings.  
*Medium.* <https://towardsdatascience.com/clip-model-and-the-importance-of-multimodal-embeddings-1c8f6b13bf72>

*Stylish product Image Dataset.* (2022, May 21). Kaggle.  
<https://www.kaggle.com/datasets/kuchhbhi/stylish-product-image-dataset>