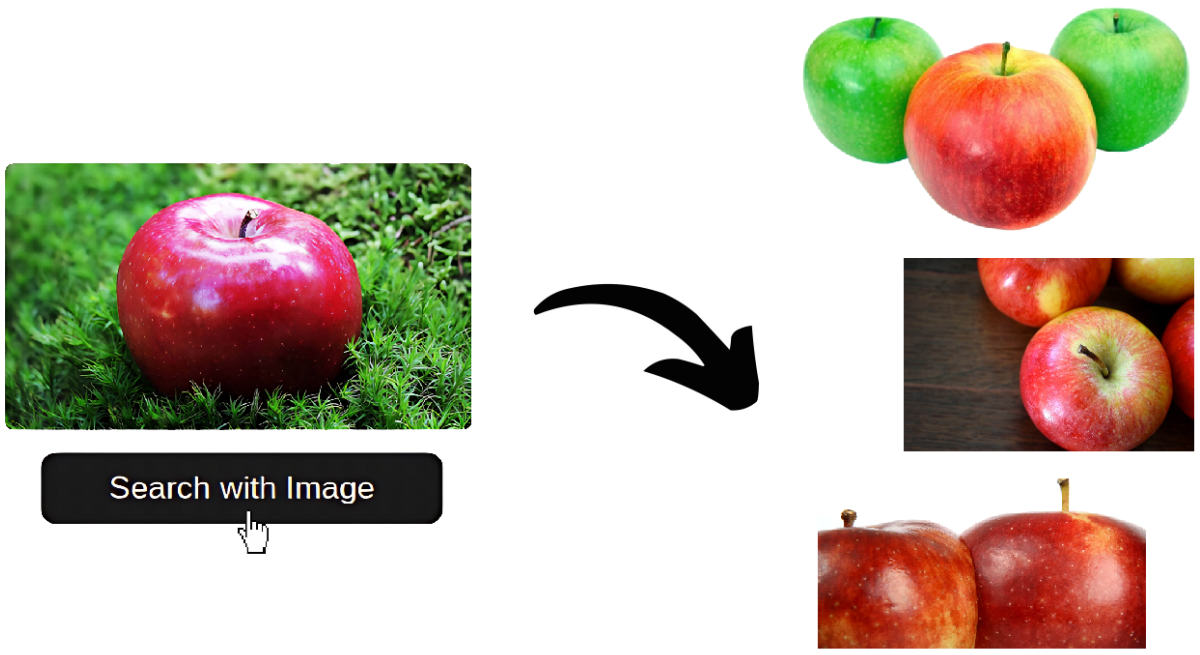
CHALMERS
UNIVERSITY OF TECHNOLOGY



Exploring Image-to-Text Visual Search Using Open Source Models

Master's thesis in Data Science & AI

TOMMY LIU

DEPARTMENT OF ELECTRICAL ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2026
www.chalmers.se

Master's thesis 2026

Exploring Image-to-Text Visual Search Using Open Source Models

TOMMY LIU

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Division of Signal Processing and Biomedical Engineering
Chalmers University of Technology
Gothenburg, Sweden 2026

Exploring Image-to-Text Visual Search Using Open Source Models
TOMMY LIU

Supervisor: Ida Häggström, Department of Electrical Engineering
Examiner: Ida Häggström, Department of Electrical Engineering
Company Supervisor: Albert Dahlin, Webbhuset

Master's Thesis 2026
Department of Electrical Engineering
Division of Signal Processing and Biomedical Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Visual searching with an image depicting an apple. Conceptual illustration, not indicative of actual results.

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2026

Exploring Image-to-Text Visual Search Using Open Source Models
TOMMY LIU
Department of Electrical Engineering
Chalmers University of Technology

Abstract

Visual searching refers to the use of visual data, typically images, in order to perform a search rather than textual input. Most visual search implementations rely on performing similarity searching over image features, in which a user-submitted query image is compared against all searchable entries' features before returning sufficiently similar results. This thesis explores a different method which utilizes image descriptions generated by vision-language models instead of image features, where the descriptions are converted into embeddings in order to match with other search entries. Evaluation data indicate that the method can provide satisfactory retrieval performance in addition to maintaining a low search query execution time, provided that an adequate vision-language model is employed and sufficient server capacity is available.

Keywords: visual search, machine learning, deep learning, embedding, vision-language model, transformer, e-commerce.

Acknowledgements

I would like to extend my gratitude to Ida Häggström, my academic supervisor and examiner. On top of your guidance, this thesis would not have been a reality without your help. I also want to thank my company supervisor Albert Dahlin, not only for sharing your vast technological knowledge, but also for the underlying idea of the thesis which originated from our discussions together. Lastly, I am grateful to Martina Johansen at Webbhuset for taking the time to speak to me about the company, the possibility of doing the thesis there and helping me secure it. Thanks to all of you.

Tommy Liu, Gothenburg, January 2026

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

AP	Average Precision
CBIR	Content-based Image Retrieval
CNN	Convolutional Neural Network
DCG	Discounted Cumulative Gain
FC (Layer)	Fully Connected (Layer)
IDCG	Ideal Discounted Cumulative Gain
mAP	Mean Average Precision
nDCG	Normalized Discounted Cumulative Gain
VLM	Vision-Language Model
WSGI	Web Server Gateway Interface

Contents

List of Acronyms **ix**

List of Figures **xiii**

List of Tables **xv**

1 Introduction **1**
1.1 Overview . . . 1
1.2 Thesis Context & Method . . . 2
1.3 Research Questions . . . 3
1.4 Related Work . . . 3
1.5 Ethical Considerations . . . 4

2 Background **7**
2.1 Convolutional Neural Network . . . 7
2.2 Object Detection . . . 8
2.3 Embeddings & Similarity Searching . . . 8
2.4 Transformer . . . 9
2.5 Vision-Language Models . . . 12
2.6 Text Embedding Models . . . 13
2.7 Vector Databases . . . 13
2.8 Image-to-Text Retrieval Method . . . 14
2.9 Evaluation . . . 14

3 Method **17**
3.1 Equipment . . . 17
3.2 Data Scraping . . . 17
3.3 Visual Search System . . . 17
3.4 Product Embedding Options . . . 18
3.5 Vision-Language Model Information . . . 18
3.6 Text Embedding Model Information . . . 19
3.7 Search Page . . . 20
3.8 Evaluation Method . . . 20

4 Results **23**
4.1 Retrieval Performance . . . 23
4.2 Qualitative Retrieval Results . . . 27

4.3 Search Time 28
4.4 Performance vs. Time 28
4.5 Analysis & Discussion 29

5 Conclusion **33**
5.1 Summary 33
5.2 Limitations 33
5.3 Future Work 34

Bibliography **37**

A Appendix 1 **I**
A.1 Model Parameters I

B Raw Search Data **III**

List of Figures

2.1 An overview of the Transformer architecture. [17], CC BY 4.0 10
2.2 Residuals depicted by red path/arrow. [17], CC BY 4.0 11
2.3 A visual summary of how attention is calculated. [19], CC BY-NC-ND 4.0 12

3.1 The search page as the user first enters. 20

4.1 The top 6 results from five different query images. Q denotes query image, while number denotes relevance score of result. 27
4.2 A plot between search time and nDCG@30 for all query images and VLMs. 29

List of Tables

4.1 Mean nDCG@30, embedding only contains name and provided description . . . 25
4.2 Mean nDCG@30, embedding only contains generated description . . . 25
4.3 Mean nDCG@30, embedding contains name, provided description and generated description . . . 25
4.4 Hit@10, embedding only contains name and provided description . . . 26
4.5 Hit@10, embedding only contains generated description . . . 26
4.6 Hit@10, embedding contains name, provided description and generated description . . . 26
4.7 The mean time it took to complete a search query for every VLM and image combination, together with the standard error. The time is in seconds. . . . 28

1

Introduction

1.1 Overview

The ability to perform a search using visual properties or the content of an image has been a desired feature for decades [1]. Conventional search methods where input text is matched with keywords, file IDs or other textual metadata had already been established, but it was not without drawbacks. Among them, one limitation was that text searches were unable to adequately match visual properties or semantic features of desired search results. Ongoing development over time has led to the widespread adoption of visual search, with Google claiming in an article published 2024 that their visual search tool Google Lens processed 20 billion searches monthly (20% of them for shopping) [2], while an article by Amazon originally posted 2024 claims a 70% year-to-year increase in visual searches [3]. Given the amount of recent work and attention that has been put on visual search tools (some just mentioned, others discussed in section 1.4), it is evident that both demand and the corresponding development to meet it are present.

Numerous example implementations have been developed over the years. Among the earliest works is IBM's Query by Image and Video Content (QBIC) system [4] in which users could search for similar images by providing color(s), a shape, or drawing a sketch with the accompanying drawing software. Another one comes from Virage [5] (old company specializing in video and rich media communications software) who created a framework for users to build their own tailor-made visual search system. As for more popular implementations, Google added their "Search by image" feature to Google Images in 2011, letting users either upload an image or a link to an image in order to search for similar ones, and later in 2017 expanded the feature with Google Lens which is also available as a smartphone application. The reverse search engine TinEye, launched in 2008, is another example. Other works include Bing [6], Pinterest [7], eBay [8], and more.

Many terms are used to describe this feature and most commonly, one will see the phrases visual search, reverse image search and content-based image retrieval (abbreviated as CBIR). Therefore it is worth clarifying what each term means. Do note that none of these have an official, standardized definition, and are instead defined based on how they are usually used. *Visual search* merely refers to using visual data (e.g., colors, shapes, or an entire image) as your search query while the output (retrieved data) can be anything (but are usually entries with associated images). *Reverse image search* commonly refers to using an image as query in order

to find its duplicates across the web. Meanwhile, *CBIR* refers to that the search mechanism retrieves images based on their visual features (hence "content-based"). Although the work in this thesis mostly falls under visual search and CBIR, the rest of this report will primarily use the former phrase, as it is common in recent related work to do so.

Implementation methods have evolved over the decades, but the underlying concept behind them has largely stayed the same. Visual searching tools primarily make use of visual features from images (or video frames). Every entry in the collection (of possible search results) has an associated image and its features stored. When the user submits the search query image, its features are extracted and a similarity search (see section 2.3) is performed between those and features stored in the database, before returning the results in an order ranked by how similar they were to the query. Meanwhile, the implementation itself has changed over time, and as of this writing, deep learning methods are the current state-of-the-art. Most commonly, a convolutional neural network (CNN) architecture is utilized because of its ability to process spatially dependent data (like images) and extract meaningful feature maps out of them.

On top of performing similarity search between visual features, many implementations add extra techniques that aid search performance. One of them is the use of image classification, which is convenient as CNNs are usually created for this purpose. This involves labeling every entry in the item collection as a particular class. When a user performs a search, a class will be determined for the query image which will narrow down the search results to entries of that particular class only. Object detection also has its place in visual search tools, allowing more precise results. Instead of using the entire image as a query, objects are detected and turned into possible search queries (the user chooses which object to use as a query). Each object is also classified into a particular class, retaining the benefits of narrowing down the search.

1.2 Thesis Context & Method

On the other hand, this thesis is about exploring a more novel method of performing visual searching. The project is done with Webbhuset, a company specializing in creating custom-tailored e-commerce solutions, which means that the search feature will be implemented in an e-commerce context where the user is able to search for products. The method still involves deep learning, but instead of working directly with image features, it will instead utilize an image-to-text process where a vision-language model (VLM) generates a description of the image, which will be used to search for other products. The purpose of the thesis is partly the exploration itself, but also potential availability that the method provides. Both time and the computational resources required for this method is significantly less than what is usually required to train one's own deep learning model(s), while being easier to implement, which would allow more developers the possibility to create a viable visual search tool. This method does have the potential drawback that search results will likely not be as precise as a system made with the image features method,

especially when compared to the more advanced implementations out there.

1.3 Research Questions

A minimum acceptable outcome and two research questions (RQs) were created to guide the work of the thesis. The former is defined as the visual search tool being able to return at least three relevant entries in the top 10 results, for three different categories of products. In addition, the search query must take, at most, one minute to complete. The reason for this minimum acceptable outcome is that the standard is low enough to easily be reached, but high enough to still be usable, even if only in a narrow or limited context. Note that "relevant" means that the entry is in the same top category as the query image (the data is distributed among different top categories and subcategories, this is further explained in chapter 3).

As for the research questions, they are as follows:

1. With the method explored in the thesis, is it possible to create an acceptable visual search tool in e-commerce using only open source models in a plug and play fashion?
2. Is it possible for a search query to be executed quickly enough for regular usage?

To expand on RQ1, "acceptable" refers to the minimum acceptable outcome above. Open source models means that the models, their code and weights are available for anyone to utilize. Plug and play fashion means that the models will not be altered in any shape or form, i.e., will not be retrained or fine-tuned. As for RQ2, quickly enough for regular usage means that the search query takes at most five seconds, the reasoning for this being that most online search tools instantly return results, and therefore five seconds should be a fair compromise. Note that five seconds is a higher standard than in the minimum acceptable outcome.

1.4 Related Work

Al-Lohibi, et al. [9] made Awjedni, a smartphone application where users can search for particular categories of clothing using pictures as input. There were 10 categories in total: ankle boots, bags, coats, dresses, pullovers, sandals, shirts, sneakers, t-shirts and trousers. Various products fitting of these categories were scraped from different sites and added to the application's database. The searching itself made use of a convolutional neural network that was trained and tested on 5025 images scraped from around the web, framed as a classification task. Once deployed, the following would happen: the user could submit an image depicting a piece of clothing, which, after pre-processing, would be sent through the trained CNN, classified into a particular category along with having its features extracted and compared to the products in the database, leading to products similar to the query image being returned.

Araujo, et al. [10] made pyCBIR, an image retrieval tool for scientific images, i.e.,

images taken during various scientific workflows, some examples being pictures depicting different materials like plastic and glass, ceramic matrix composites, nanostructural features of thin films, etc. The paper tried out two different well-known CNN architectures, LeNet and Inception-ResNet-v2, training them on the database data and using them during the searching process. When a user submits a query image, the features are extracted via the CNN and compared to database's images via a similarity search, returning entries that are similar enough.

Shop The Look, the name behind one of Pinterest's visual search applications by Shiau, et al. [11] is more advanced. Shop The Look's search process begins by performing object detection over the query image and identifying possible query objects. Every localized object is turned into its own query image, and the user can see each object's search results by tapping on it. The object is also classified into a particular category, which restricts the searching only to that class in order to reduce the amount of irrelevant items retrieved. The object detection architecture they used was ResNext101-FasterRCNN [12] and the backbone network for extracting visual embeddings was ResNext101-32x8d [12].

From Yang, et al. [8], eBay's visual search system uses a modified ResNet-50 [13] model which not only classifies images into particular categories, but also generates a binary hash out of their feature vectors which significantly reduces storage and retrieval computation costs. When the user submits a query image, it is not only classified, but a binary hash is also created which is used as the query during similarity search. On top of that, they train multiple gradient-boosted models in order to predict aspects of products, e.g., color, brand, material, etc. These predictions are combined with the similarity searching in order to rank results with more precision.

The CNN architecture in particular is not required as Yada, et al. [14] proves. Their method is conceptually the same (performing similarity search over visual features), but instead of a convolutional neural network, the authors make use of SigLIP [15], a vision-language model, which was fine-tuned on image-title pairs from their e-commerce website Mercari. The resulting image encoder was used to produce the image feature vectors that CNNs usually produce.

1.5 Ethical Considerations

Although the work done in this thesis does not involve fine-tuning or model training, any future work based on this method may consider such options. In a real-world context, the developer(s) may choose to save user-submitted query images as training data for further fine-tuning, which brings its set of risks as the images might contain sensitive information. Even if not included in any training set, the act of storing the images itself raises questions about privacy risks as well, especially considering that there likely will be imprudent users submitting sensitive images.

As using a photo to shop for products might, in certain contexts, be more convenient than text searching, it could potentially lead to more completed purchases compared to if the visual search tool were not available for customers to use. This raises ethical questions about the convenience of engaging in online shopping activities,

especially for people who enjoy shopping for its own sake and even more so for individuals whose excessive spending is causing problems in other areas of their life. An argument can be made that convenient shopping tools can cause significant negative impact on people with shopping addiction, to the point that these tools should not exist.

2
Background

This chapter will detail all the technical background that underlie the topic of the thesis.

2.1 Convolutional Neural Network

As the convolutional neural network is a popular choice for visual searching, on top of being widely used in multiple computer vision tasks, an introduction to this particular architecture is appropriate. The CNN is a type of artificial neural network designed to learn and predict spatially dependent data, like images. One of their central aspects is the convolutional layer, in which a sliding window called the kernel is moved throughout the data, performing an operation called convolution at each patch. This creates a feature map containing representations of the data, which are fed into further layers of the network. In earlier layers, the feature maps capture smaller details, while later, the maps contain more complex, semantic structures. As an example, if the data is a picture of a cat, the earlier layers may capture details like edges, while later on, it may be the cat's eyes, ears, and eventually the cat as a whole.

The kernel size is usually 3x3 or 5x5, but can be any size (including asymmetrical dimensions like 1x5) if compatible with the data's dimensions. Each tile in the kernel corresponds to a value, called a weight. The convolution operation is essentially the sum of an element-wise multiplication between the kernel weights and the overlapping data patch's values. This is repeated as the kernel slides throughout the data, usually starting at the top-left and ending at the bottom-right. The result of these calculations form the feature map, which is sent as an input to the next layer where the process repeats.

In the end, the feature maps are utilized for whatever purpose the developer has in mind. For classification tasks, the last layers of the CNN are usually fully connected layers which use these maps to classify their corresponding data. In object detection (see section 2.2), the feature maps are processed in order to localize objects and classify them. Within visual searching, these feature maps are the basis on which similarity search is performed.

2.2 Object Detection

Object detection is when a machine learning model detects specific objects or entities on an image, localizes them (i.e., identifies their position in the image) and classifies the entity. For example, when an image depicting dogs and cats is sent to an object detection model, it will draw boxes around the dogs and cats and classify each box appropriately. Object detection models are usually structured with a backbone, neck and head and an explanation of each will follow shortly.

The backbone is responsible for extracting feature maps from the input image which contains information on both lower-level details like edges and textures and higher-level concepts like entire objects in the image. This is usually handled by either a CNN or a transformer (see section 2.4). Transfer learning (where a model or parts of the model that were trained for another task is repurposed for something else) is suitable here. For example, Faster R-CNN, an object detection model made by Ren, et al. [16] uses ResNet-101 [13], a CNN with residual connections, as its backbone.

Following the backbone is the neck. This part is responsible for fusing all feature maps together, which helps the detector find objects at different scales. The neck is optional, but without it, the model ends up less capable of detecting smaller objects and processing images containing objects of varying sizes. A common neck model is the Feature Pyramid Network (FPN) where the highest-level, lowest resolution feature map is essentially added to the next level down. This repeats until all the maps are added together. The map that is next level down is first sent through a convolutional layer with kernel size 1x1 before being added to the level above.

At last, the head does the actual detecting and classifying itself. It is responsible for localizing objects and determining bounding boxes for them before applying a classification that tells us what the object is. There are primarily two types of heads, one-stage detectors and two-stage detectors, with the difference being that one-stage detectors will immediately predict bounding boxes and classes after receiving the feature map, while the two-stage counterpart will first propose regions in which objects may exist, examining them more closely before deciding where the objects exist and what class they belong to. One-stage detectors usually have faster inference times, while two-stage detectors are generally more accurate.

Its use in visual searching is to help narrow down focal points in a query image. For example, if the query depicts an jacket, but also a desk lamp to the right and a chair to the left, using the image as a whole to search may return results pertaining to any of these objects, or the scene as a whole. This might be desirable in certain contexts, e.g., when the user wants the results to match the entire scene, but if the user wants the focus to be on a certain object, the detection model can be used to localize and isolate objects to help narrow down the focus.

2.3 Embeddings & Similarity Searching

Embeddings are numerical vector representations of data (text, images, etc.) which capture numerous features and properties of whatever they represent (e.g., for a text

sentence, an embedding might encapsulate the grammatical structure, meaning and relations between words). The purpose of embeddings is not only because machine learning models are better able to process numerical vectors compared to the data's original form, but also because of the feature capturing ability which leads to use cases like finding similar data based on features that are difficult to describe in words or to quantify. The reason all of this works is because embeddings are projected into a common vector space where position, distances (relative to each other) and directions represent different semantics and meanings.

The act of finding similar entities via embeddings or feature vectors is called similarity searching. This is done by converting the query data into a query embedding (this is comparable to a search query) and calculating the distance between this embedding and all other embeddings in the vector space. There are several distance metrics that can be used, but the most common ones are cosine similarity and Euclidean distance, where the former measures the angle between vectors while the latter measures the straight-line distance. The closer the embeddings are, the more similar they are considered.

2.4 Transformer

Since the transformer architecture [18] underlies a lot of the models used in this thesis, a description of them will be included. The transformer is a neural network architecture that takes sequential data as its input and contextualizes it. The main mechanism utilized is called attention, which essentially entails computing how much the individual components of the data (tokens) correspond and relate to each other. Tokens themselves can take on any form depending on the type of input data and the developer's choice, e.g., individual characters, sub-words, words, image patches, etc. Transformers typically consist of an encoder and a decoder, where the encoder converts the input sequence into a learned representation packed with contextual information, while the decoder uses the learned representation in order to generate data with respect to a particular task (e.g., language translation, predicting the next word in a text sequence, etc.), although some models only make use of the encoder or the decoder depending on its purpose and function. Figure 2.1 shows the architecture's overall structure.

Before the input data even enters the encoder, it will be transformed into a vector representation, i.e., an embedding. The reason for this is because the model cannot meaningfully process raw input data the same way it does for numerical vectors, and allows for linear algebraic operations like matrix multiplication to be utilized. On top of that, positional encodings are created, based on a collection of sine and cosine functions. These represent each token's position or order in the sequence, and are added to the corresponding embedding, creating positional embeddings.

Once an embedding for each token is created, they are sent into the encoder, which consists of the multi-head self-attention layer and the feedforward network. In-between layers, the data will also be normalized and have its residuals added to it (residuals being the exact same data but before it was passed through the layer).

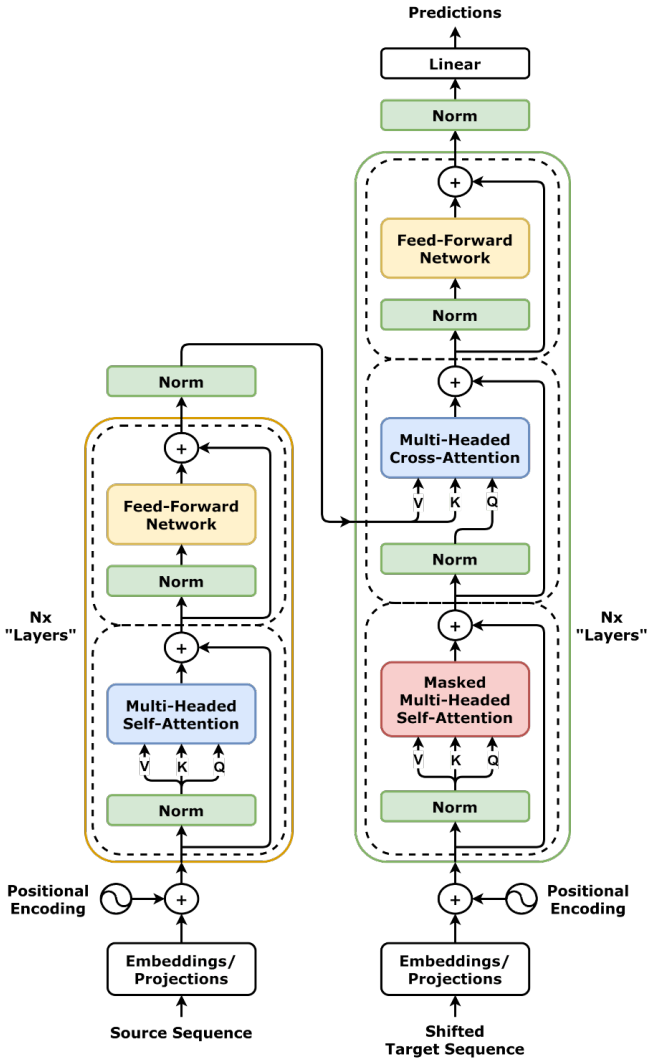


Figure 2.1: An overview of the Transformer architecture. [17], CC BY 4.0

Figure 2.2 is a zoomed-in version of the architecture overview showing one of the residuals with a red colored arrow. This particular example depicts the residuals at the beginning of the encoder, but similar can be seen at other parts of the architecture.

The multi-head self-attention layer is where the attention mechanism is employed, where each token's relation to other tokens is calculated. This is done via queries, keys and values, each head containing their own weight matrices for them. The query, key and value matrices are multiplied with each token's embedding in order to create query, key and value vectors. For each given token, the query vector represents what the token is "asking" for, the key vector represents what it offers (e.g., grammatical structure, meaning, etc.), while the value vector contains the contents of the offer itself. The query and key vectors are multiplied together with a dot product, creating attention scores which are divided by the square root of the query/key vector dimension, and sent through a softmax function before being multiplied with the value vector. This creates the output vector which is a representation of the

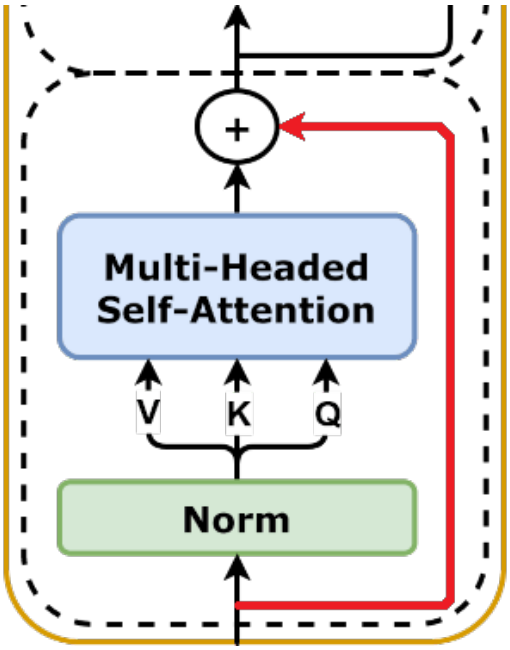


Figure 2.2: Residuals depicted by red path/arrow. [17], CC BY 4.0

given token, but with contextual information added.

An overview of the attention calculation process can be seen in figure 2.3. The top-left panel illustrates the multiplication of positional embedding with query, key and value matrices, creating their respective vector. X stands for the embedding, W^Q, W^K and W^V represent the query, key and value matrices respectively while Q, K and V refer to query, key and value vectors. The bottom panel shows the equation for calculating the output vector Z, where K^T is a transpose of the key vector and d_k is the query/key vector dimension. The top-right panel depicts the equation calculation in a flow-chart manner.

After another round of normalization and residual addition, the data reaches the feedforward neural network, usually consisting of a fully connected (FC) layer, a ReLU activation function, and another FC layer. The first FC layer expands the dimension of the output embedding and mixes its features together, its purpose being to find further insights within an embedding. The ReLU activation function allows the model to learn non-linear, more complex patterns. The last FC layer compresses the data back to its original dimension, keeping the most important insights it gained from the expansion.

The decoder's responsibility is to generate data based on the ground truth data and the encoder's outputs. During training, the target data is turned into positional embeddings (just like in the encoder), before being sent to the masked multi-head self-attention layer. This is largely the same compared to the encoder, except that tokens cannot attend to any other tokens that come later in the sequence, only to itself or tokens that come before. As the decoder generates new data based on what it has currently seen and generated, the purpose for the mask is for the model to not "cheat" by looking ahead.

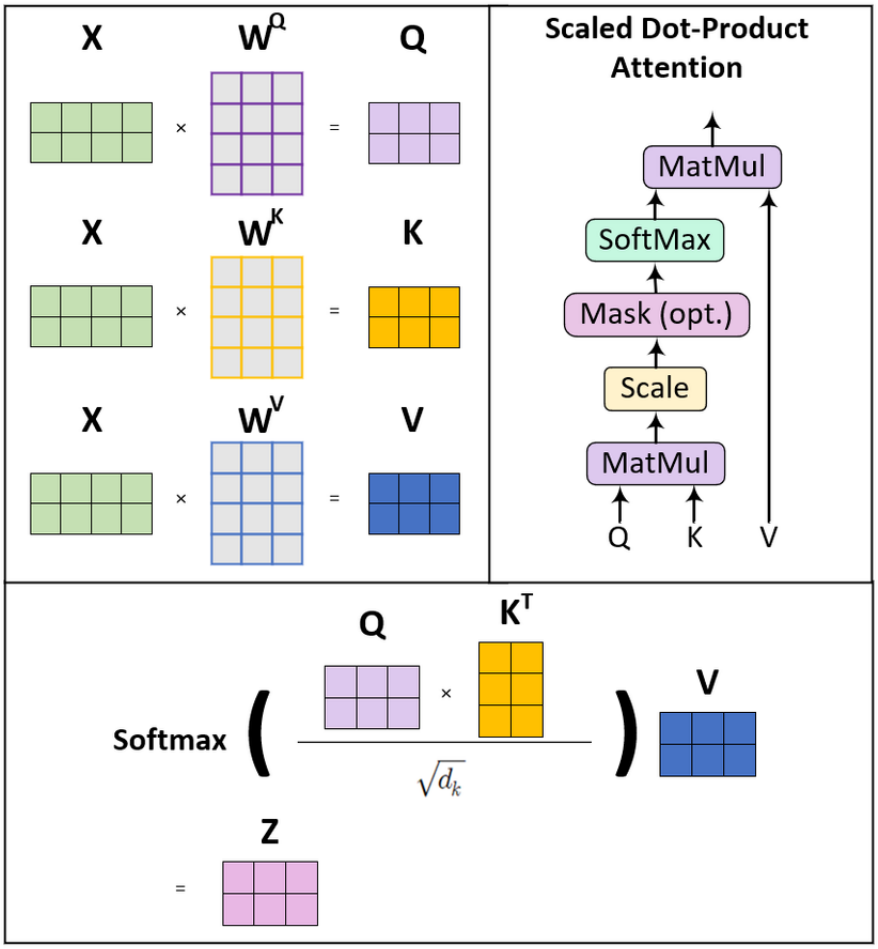


Figure 2.3: A visual summary of how attention is calculated. [19], CC BY-NC-ND 4.0

The multi-head cross-attention layer follows afterward. Here, the encoder's outputs are received, together with the ground truth token embeddings (after processing from the previous layer). In contrast to a regular multi-head self-attention layer, the query vectors are created from the ground truth tokens, while the key and value vectors are calculated from the encoder outputs. This is done to make sure the generated output is contextually relevant to the inputs. A feed-forward network follows which works similarly as in the encoder, before being sent to an FC layer and a softmax function. As a result, each possible token receives a probability for being the next one in the sequence. These probabilities are compared to the next ground truth token before the model's weights are appropriately updated.

2.5 Vision-Language Models

Recall that the method in this thesis uses generated image descriptions as the basis for similarity searching, in which vision-language models (VLMs) are responsible for creating. VLMs are generative models that combine visual and textual capabilities

and are trained to generalize their capabilities to multiple tasks, like image captioning, visual question and answering, optical character recognition, and more. Their input consist of images, and for more advanced models, text prompts that can guide its efforts toward what the user wants, while their output is usually text.

VLMs vary in their exact architecture, but there are some commonalities between all of them, that being the presence of a vision/image encoder, a text decoder and a means of mapping visual and textual data together (cross-modal fusion). Both the encoder and decoder structure are similar to what was presented in section 2.4, with some possible variation depending on the model. Instead of tokenized text, any input image will be divided into patches before being sent into the encoder. Beyond that, architectures tend to differ. One of the simpler models comes from Wang, et al. [20], who presented Generative Image-to-text Transformer (GIT) which only contains an image encoder and text decoder. The encoder's outputs are concatenated with the text decoder's input in order to perform cross-modal fusion (the paper also mentions using cross-attention instead, but states that performance was worse). Meanwhile, Microsoft's Florence-2 from Xiao, et al. [21] used DaViT [22] as their vision encoder coupled with a multimodal encoder & decoder. The output of the DaViT is concatenated with the user's text prompt, which is sent into the multimodal transformer whose decoder consequently generates the textual output.

2.6 Text Embedding Models

Performing similarity searching with textual data requires the text first be converted to an embedding, which is where text embedding models come in. Quite simply, these models take a particular string of text as input and generate a vector representation of it, which can then be used to compare to other embeddings. Its architecture mainly consists of transformer encoder layers, with some popular open source models having 12 of them in succession.

2.7 Vector Databases

Traditional databases like MySQL, PostgreSQL, etc. store their data in tables with rows and columns where the former often represent a particular entry, while the latter denote a particular field or attribute. Retrieval tends to be precise, matching exactly what the user has queried, whether it'd be an exact entry from an exact table, every entry with an id below or above 50, etc. In contrast, the main purpose of a vector database is to store vector embeddings, along with any other fields that the user may want. Querying is generally done by running a similarity search between a query embedding and other embeddings in the database, making them match on semantic features rather than precise criteria. This makes vector databases useful within retrieval systems where user-desired items are not easily identified by objective measures, or where the desired features of the items are either too complicated to describe or store in a traditional database, let alone being queried.

2.8 Image-to-Text Retrieval Method

Much of what is covered so far culminates into the method used in this thesis. Specific implementation details relevant to the project can be found in chapter 3, but what follows is a general description over the method.

Before any searching can even be done, the developer needs to set up a vector database with an entry for each retrievable entry. The most important field for each entry is its associated embedding, which the query embedding will match on later when the user performs a search. The embedding will be created using the text embedding model of choice, and the exact textual data that it will be based on depends on what the developer wants. If the search entry has a name, description, tags, keywords, etc. (which tends to be the case in e-commerce contexts), the simplest approach is to concatenate them together and create an embedding out of it. If the search entries contain associated images, then another approach is to create the embedding out of VLM-generated image descriptions. As long as the textual data is actually relevant to the search entry, the embedding should be adequate.

As for the search procedure itself, it begins with the user uploading a query image. This image is sent to the VLM of choice, which generates a description. This description is then converted into an embedding using the same text embedding model as when the database was created. A similarity search is performed between the query embedding and all other embeddings in the database, with the results returned in an order that matches the calculated distances.

2.9 Evaluation

Evaluating the performance of a search tool does not have an obvious process as it needs to model what its intended users want out of it. In most of the literature, the authors' data (both query images and potential results) are labeled, each belonging to a particular category. They can therefore deem a particular search result relevant if it belongs to the same category as the query image. An overview of typical evaluation metrics and methods will be presented.

In the context of a search tool, precision refers to the proportion of returned results that truly are relevant. Its equation looks like the following:

$$\text{Precision} = \frac{TP}{TP + FP}$$

where TP and FP means true positives and false positives, respectively. In this context, a true positive is a retrieved result that was truly relevant (to the query image) while a false positive is a retrieved result that was not.

Recall refers to the proportion of all possible relevant results that were retrieved. In other words, of all the relevant entries, what proportion was returned? Its equation is as follows:

$$\text{Recall} = \frac{TP}{TP + FN}$$

where TP means true positives, and FN means false negatives. The latter refers to relevant entries that were not retrieved.

Although these metrics in their original form are not popular in the visual search literature, variations of them are commonly used. Recall@K is an example, where it only evaluates recall based on the top K results and ignores the rest. As an example, say that K = 20 and there are 30 relevant items. If the top 20 results contain 15 relevant items, that gives us:

$$\frac{15}{30} = 0.5$$

or 50% as the Recall@K. Note that true positives beyond the top 20 are ignored, therefore Recall@K cannot be 100% in this scenario as not all 30 relevant entries can fit into the top 20. This is something to be wary of if using this metric. Precision@K is conceptually similar, except that it measures precision instead.

Average precision (AP) is another common evaluation metric in information retrieval systems, which not only takes into account the relevance of retrieved entries, but also the order in which they are returned. The gist of how average precision is calculated is by sequentially going down the list of retrieved results and for each position, calculate the precision of entries at the current position and everything previous. For example, if the calculation is currently at position 3, then the precision will be calculated for entries at position three, two and one. The following is the equation:

$$\text{AP} = \frac{\sum_{i=1}^{n} P(i) * rel(i)}{\text{amount of relevant documents}}$$

where n is the amount of retrieved search results, i is the position, $P(i)$ is the precision at position i, and $rel(i)$ is a binary function representing if the entry at position i is relevant or not, returning a 1 if relevant and 0 otherwise. This calculation is done for every search query, and when multiple search queries are evaluated in the aggregate, mean average precision (mAP) is used instead, which is simply the mean of all APs.

Although mAP takes relevance into account, it is only judged on a binary basis, i.e., a retrieved entry can either be relevant or not, with nothing in-between. If a graded relevance scoring system is desired instead, one can use discounted cumulative gain (DCG), which is another evaluation metric whose scoring is based on both order and relevance. There are a few variants on the equation, but the following is one of them:

$$\text{DCG@}K = \sum_{i=1}^{K} \frac{2^{rel_i} - 1}{\log_2(i+1)}$$

where K is the amount of retrieved results, i is the current position and rel_i is the relevance score at position i. The equation essentially sums up all the relevance

scores for a given search result, but discounts any scores that appear later in the sum which penalizes the search tool for retrieving relevant entries later than they ideally should be retrieved.

As DCG may vary wildly in its distribution, even within the same search tool and data due to differences in retrieved entries, the amount of entries in each category, etc. this makes it difficult to compare DCG between different search queries. To solve this problem, normalized discounted cumulative gain (nDCG) is used instead, where every DCG is normalized based on the "ideal" result, also called the ideal discounted cumulative gain (IDCG):

$$\text{IDCG@}K = \sum_{i=1}^{K} \frac{2^{rel_i^\star} - 1}{\log_2(i+1)}$$

where the only difference with DCG is the presence of $rel_i^\star$, which represents the relevance score of the "ideal" entry at position i.

Together, they create nDCG through the following:

$$\text{nDCG@}K = \frac{\text{DCG@}K}{\text{IDCG@}K}$$

which normalizes the total score to a value between 0 and 1, and makes it comparable to other queries.

However, a common use case with search tools is to find one precise result, in contrast to a type or group of results. For example, one might be trying to find a certain brand and color of a jacket, instead of browsing for jackets overall. Evaluating this particular aspect can still be done with nDCG, especially if proper relevance scores are assigned, but if the developer wants to directly assess this facet, Hit@K is simpler to use. For a given search query, this metric deems the output a hit if at least one relevant entry appeared in the search results, and a miss otherwise. The metric is subsequently calculated based on the proportion of queries that produced a hit:

$$\text{Hit@}K = \frac{\text{Hits}}{\text{Queries}}$$

But ultimately, an information retrieval needs to be useful for the target audience that is going to use it and user feedback can be a valuable tool for adjusting the search tool. For example, Shiau et al. [11], in Pinterest's Shop the Look system, use a metric they called End-to-end (E2E) Relevance@5, where users are able to rate the top 5 results for a given query with the options "Extremely Similar", "Similar", "Marginally Similar", "Not Similar" and "Did Not Load".

3
Method

3.1 Equipment

The following hardware equipment was used:

- GPU: Nvidia Geforce RTX 4060
- CPU: 13th Gen Intel(R) Core(TM) i3-13100

As for the software:

- Ubuntu 22.04.5 LTS
- Python v3.10.12
- HTML
- CSS
- JavaScript

3.2 Data Scraping

The dataset will be detailed in chapter 4, but it is worth noting that its structure consists of top categories with subcategories within each of them. As for its acquisition, the data was scraped off the customer's website. This was accomplished by inspecting browser network traffic to retrieve a JSON file containing product metadata like name, description, category IDs, etc. Data processing followed, leaving only desired categories while removing extraneous fields that were not relevant for development. Other processing measures were taken, one of them being the top category/subcategory structure. Top categories were created manually, while most subcategories were inherited from the category IDs in the original data. In addition, certain product categories would be split into different subcategories if key differences between subgroups existed. These measures were taken to test if the visual search system could correctly rank products based on finer details.

3.3 Visual Search System

The vector database used was Milvus Lite [23] which provides its own functions for querying the database, either though similarity searching or otherwise, and those

functions were used to perform the searching itself. When creating the database, the developer can make various choices like data index types and distance metric. For the former, FLAT was chosen. As for the distance metric, since the embeddings were floating point vectors, the available metrics in Milvus Lite were cosine similarity, Euclidean distance and inner product, but because the embeddings were also normalized, the choice between those three did not matter in terms of similarity search results. Ultimately, cosine similarity was selected, and its equation is as follows:

$$\text{Cosine similarity} = \frac{\mathbf{A} \cdot \mathbf{B}}{\| \mathbf{A} \| \| \mathbf{B} \|}$$

where $\mathbf{A}$ and $\mathbf{B}$ represent vectors or embeddings. The dot $\cdot$ stands for dot product and the parallel lines $\|\|$ represent the magnitude of the vector inside.

The searching script itself consisted of choosing a query image, sending it through a chosen VLM to generate a description which was then provided to the same text embedding model used during database creation to produce the query embedding. Milvus's own similarity search function was then utilized to conduct the search, retrieving the results ranked with respect to the calculated distances.

3.4 Product Embedding Options

During database creation, a choice to make is what the products' embeddings will be based on, i.e. exactly what text should be converted into an embedding in order to represent the particular product. Here, three different options were tested:

- Option 1: The embedding is only based on the provided name and description from the scraped data.
- Option 2: The embedding is only based on a VLM-generated product description, generated from the provided product image.
- Option 3: The embedding is based on provided name and description, together with a VLM-generated product description.

During the experiments, if a certain VLM was used to generate product descriptions, the same model was also used for the query images. In reality, this does not have to be the case (more in chapter 5).

3.5 Vision-Language Model Information

The following is a list of the VLMs used in the thesis:

- Florence-2-large-hf [21]
- SmolVLM-256M-Instruct [24]
- moondream2 [25]
- blip-image-captioning-base [26]

- blip2-opt-2.7b [27]
- git-base-coco [20]

The first three VLMs tend to generate more detailed descriptions, while the latter three generate shorter captions. This is due to a combination of how they were configured and how they were trained.

Speaking of their training, the type of data they were trained on were predominantly image-text pairs, sometimes containing other types like visual Q&A pairs, region captions (describing only a part of an image), etc. For example, Florence-2-large-hf was trained on FLD-5B, a dataset that the model developers themselves assembled [21] which not only contains image-caption pairs, but also bounding boxes and segmentations to delineate regions, captions for each region, but also combinations of regions, etc.

As for the task(s) they were trained on, this differs across models. But one common task among most of them was language modeling, in which the model predicts the next word/token given the previous words and input. In this case, the input is an image, but can also contain a text prompt to guide the output generation. Certain models were trained on other tasks as well, like image-text contrastive learning where images and text are mapped onto a feature space, with appropriate texts placed closer to corresponding images while unsuitable texts and images are pushed further apart.

Any relevant parameters or model settings will be listed in the appendix, section A.1.

3.6 Text Embedding Model Information

The following is a list of text embedding models used, along with the dimension of the embeddings they produce:

- all-MiniLM-L12-v2 [28] - dim. 384
- all-mpnet-base-v2 [29] - dim. 768
- bge-base-en-v1.5 [30] [31] - dim. 768
- bge-large-en-v1.5 [30] [31] - dim. 1024
- bge-small-en-v1.5 [30] [31] - dim. 384

All of these models were either pretrained on large amounts of text to perform language modeling, or built upon a pretrained backbone with similar data and task. The models were then fine-tuned on contrastive learning using vast amounts of sentence pairs, i.e. mapping positive (similar) sentence pairs close to each other on a feature space while pushing apart negative (dissimilar) pairs. To clarify, this fine-tuning was done by the developers of the model and not specifically for this thesis. The models were all trained on English data and thus only work on that language.

3.7 Search Page

A search page was also created in order to produce the same experience as if the user was searching on a real website. As mentioned before, the code was written in HTML, CSS and JavaScript. The website allows for both visual searching and conventional text searching. When the search button is pressed, a request is sent to the backend which handles the searching itself, before returning the results back to the frontend, displaying the results in their appropriate order. The backend itself was implemented in Python using Flask, a Web Server Gateway Interface (WSGI) that allows web servers to communicate with Python applications.

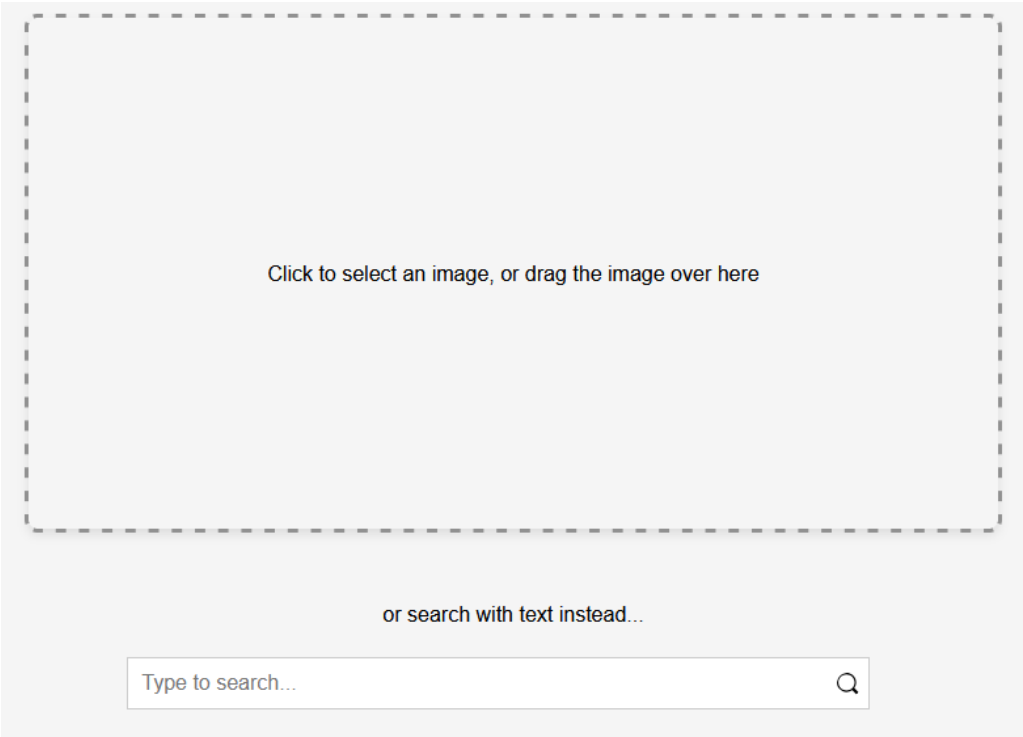


Figure 3.1: The search page as the user first enters.

3.8 Evaluation Method

In order to evaluate the performance, a set of query images were prepared. It was manually annotated and structured in the same manner as the dataset. A subset of the query images were not only used to evaluate general performance, but also to see if the search feature could find an exact match. These images in question would therefore depict a very similar, or the exact product as their intended target.

Evaluation metrics were chosen with respect to what a prospective user would want from the tool. nDCG@30 and Hit@10 were the metrics of choice, where the former evaluates general performance and whether a user who merely wants to browse similar items to their queries would be able to find them, while the latter was used to evaluate whether the user could find an exact (or extremely similar) match with

respect to their query image. K=30 was chosen for nDCG@K which is based on the amount of returned entries that tend to appear on a page, while K=10 was chosen for Hit@K in order to reward the system for returning the target product early. Within the context of nDCG, a given result would earn a 2 as its relevance score if it was in the same top and subcategory as the query image, a 1 if only the top category matched, and 0 points if it did not match whatsoever. The ground-truth ranking for nDCG is anything that achieves the IDCG, where all score 2 products appear first, followed by score 1 products, with the rest being irrelevant entries. The exact order within score 2 or score 1 products does not matter and as long as the latter come after the former, the IDCG is achieved.

A Python script was written for each evaluation metric where the script goes through every query image, performs a visual search with the same process as described above, and evaluates the search with respect to the metric. Once the script has gone through every query image, the mean for that metric is calculated. Note that the evaluation for Hit@K only runs through the subset of query images used to test for exact matches, while the evaluation for nDCG runs through all query images.

As for evaluating the time taken to perform a search, five pictures from the query set at various file sizes and resolutions were specifically chosen for this purpose, to see if and how these image attributes affect the search time. From there, each image was tested with each VLM, and for any given image/VLM combination, five searches were performed, yielding five different search query times. The mean of the five search times was calculated to represent the time required of the image/VLM combination, and the standard error was calculated as well to represent the uncertainty of the mean.

Testing was done using the search page, with the start and end of a search query defined as from the point where the user clicks the "Search with Image" button, to when the results are returned, respectively. The time itself was calculated via a timer implemented in JavaScript that adheres to the definition. Initial testing showed that the choice of embedding model and option did not significantly alter the time, or at all, so only the vision-language models were fully tested. For all tests, embedding option 1 was utilized together with text embedding model all-mpnet-base-v2. Note that testing was performed on the hardware equipment described at the beginning of this chapter (section 3.1).

The trade-off between retrieval performance and search time was also explored, and to do this, search times for all query images were measured. But each VLM and image combination was only measured once (as opposed to five times), and instead of involving the search page, testing was strictly based on the search mechanism alone. In other words, the measured time was simply between the start and end of the search process itself, without the extra steps required to process website-specific details. Once again, all VLMs were used, embedding option remained the first one and text embedding model all-mpnet-base-v2, but the timer was implemented in Python instead of JavaScript. In the end, the data was plotted with nDCG@30 data on a scatter plot in order to see the trade-off between search time and performance (which can be seen in section 4.3). Specifically, the nDCG@30 data collected with the same embedding option and model was used here.

4

Results

The following chapter will present evaluation results for the visual search system, which was evaluated on two criteria. The first is the system's ability to return relevant products given the query image. The second is the time it takes to execute a search query. Qualitative results are also included, along with a scatter plot depicting the trade-off between retrieval performance and search time.

4.1 Retrieval Performance

The dataset used was a subset of one of Webbhuset's customer's product catalogues. After data processing, it consisted of 203 products which were distributed into a number of top categories, and further divided by subcategories which were denoted by an ID. The following is a list of top categories in the dataset:

- Angle grinders
- Boots
- Eye protection (glasses, goggles and visors) - 3 subcategories
- Hammers
- Helmets
- Jackets - 3 subcategories
- Knives - 4 subcategories
- Notebooks
- Paintbrushes
- Pens - 2 subcategories
- Tape - 5 subcategories
- Trousers - 2 subcategories

The query set, being the set of query images used to test the visual search system, contained 96 images in total. Some of those images were taken by the author with a smartphone camera. There was at least three query images for each subcategory. On top of that, 11 of the query images were specifically prepared to test whether the system could help the user find an exact match or an extremely similar product. Therefore, these particular images had to represent the exact product it was intended to match or something very close.

As for the results themselves, tables 4.1, 4.2 and 4.3 illustrate nDCG@30 results, each table representing a different embedding option. Meanwhile, tables 4.4, 4.5 and 4.6 contain Hit@10 results. Each row represents choice of VLM, while column represents text embedding model.

nDCG@30 - Option 1	**Embedding**				
VLM	all-MiniLM-L12-v2	all-mpnet-base-v2	bge-base-en-v1.5	bge-large-en-v1.5	bge-small-en-v1.5
Florence-2-large-hf	81.4%	**82.9%**	78.9%	75.9%	78.7%
SmolVLM-256M-Instruct	76.7%	77.3%	74.3%	72.6%	76.6%
moondream2	80.8%	82.2%	79.9%	75.7%	80.1%
blip-image-captioning-base	71.9%	73.5%	68.4%	68.9%	67.8%
blip2-opt-2.7b	79.8%	81.2%	77.4%	76.6%	74.4%
git-base-coco	70%	69.2%	65.2%	66%	68.1%

Table 4.1: Mean nDCG@30, embedding only contains name and provided description

nDCG@30 - Option 2	**Embedding**				
VLM	all-MiniLM-L12-v2	all-mpnet-base-v2	bge-base-en-v1.5	bge-large-en-v1.5	bge-small-en-v1.5
Florence-2-large-hf	83.8%	83.6%	83.5%	**85.1%**	83.6%
SmolVLM-256M-Instruct	78.6%	76.6%	75.1%	75.9%	75.2%
moondream2	83.1%	83.5%	83.6%	82.9%	83.5%
blip-image-captioning-base	70.7%	70.8%	68.6%	70.8%	68.9%
blip2-opt-2.7b	76.7%	74.4%	73.4%	77.2%	74.5%
git-base-coco	63.1%	61%	58.1%	59.2%	60%

Table 4.2: Mean nDCG@30, embedding only contains generated description

nDCG@30 - Option 3	**Embedding**				
VLM	all-MiniLM-L12-v2	all-mpnet-base-v2	bge-base-en-v1.5	bge-large-en-v1.5	bge-small-en-v1.5
Florence-2-large-hf	83.2%	85.8%	83.9%	84.9%	81.9%
SmolVLM-256M-Instruct	79.6%	77.5%	77.1%	77.6%	77.4%
moondream2	83.9%	85.2%	84.6%	84.3%	**85.9%**
blip-image-captioning-base	74.4%	75.4%	71.7%	73.8%	71.2%
blip2-opt-2.7b	82.4%	83%	77.9%	80.5%	77.9%
git-base-coco	73.1%	71.2%	67.7%	70.6%	71.4%

Table 4.3: Mean nDCG@30, embedding contains name, provided description and generated description

Hit@10 - Option 1	**Embedding**				
VLM	all-MiniLM-L12-v2	all-mpnet-base-v2	bge-base-en-v1.5	bge-large-en-v1.5	bge-small-en-v1.5
Florence-2-large-hf	72.7%	72.7%	81.8%	63.6%	63.6%
SmolVLM-256M-Instruct	63.6%	81.8%	63.6%	54.5%	63.6%
moondream2	72.7%	72.7%	81.8%	72.7%	81.8%
blip-image-captioning-base	63.6%	72.7%	63.6%	54.5%	54.5%
blip2-opt-2.7b	72.7%	**100%**	90.9%	81.8%	90.9%
git-base-coco	27.3%	45.5%	36.4%	36.4%	45.5%

Table 4.4: Hit@10, embedding only contains name and provided description

Hit@10 - Option 2	**Embedding**				
VLM	all-MiniLM-L12-v2	all-mpnet-base-v2	bge-base-en-v1.5	bge-large-en-v1.5	bge-small-en-v1.5
Florence-2-large-hf	90.9%	90.9%	90.9%	**100%**	90.9%
SmolVLM-256M-Instruct	63.6%	54.5%	45.5%	54.5%	54.5%
moondream2	90.9%	**100%**	**100%**	**100%**	**100%**
blip-image-captioning-base	72.7%	81.8%	81.8%	81.8%	81.8%
blip2-opt-2.7b	**100%**	**100%**	90.9%	**100%**	**100%**
git-base-coco	54.5%	54.5%	54.5%	54.5%	54.5%

Table 4.5: Hit@10, embedding only contains generated description

Hit@10 - Option 3	**Embedding**				
VLM	all-MiniLM-L12-v2	all-mpnet-base-v2	bge-base-en-v1.5	bge-large-en-v1.5	bge-small-en-v1.5
Florence-2-large-hf	72.7%	81.8%	81.8%	90.9%	72.7%
SmolVLM-256M-Instruct	72.7%	63.6%	54.5%	72.7%	81.8%
moondream2	72.7%	54.5%	81.8%	90.9%	90.9%
blip-image-captioning-base	81.8%	81.8%	81.8%	81.8%	81.8%
blip2-opt-2.7b	**100%**	**100%**	**100%**	**100%**	**100%**
git-base-coco	36.4%	54.5%	54.5%	63.6%	72.7%

Table 4.6: Hit@10, embedding contains name, provided description and generated description

4.2 Qualitative Retrieval Results

Figure 4.1 shows the top 6 results from five different query images. The first image of every row is the query, and this is denoted by the letter Q above those pictures. Each of the result images/products has a number above them reflecting the relevance score of that entry. Recall that a score of 2 meant the result shared the same top and subcategory as the query, while a score of 1 meant only the top category was the same. A score of 0 meant the query and product are unrelated, at least within the categorization used in the thesis. For these results, the vision-language model used was Florence-2-large-hf, the text embedding model was all-mpnet-base-v2, and the 1st embedding option was utilized.

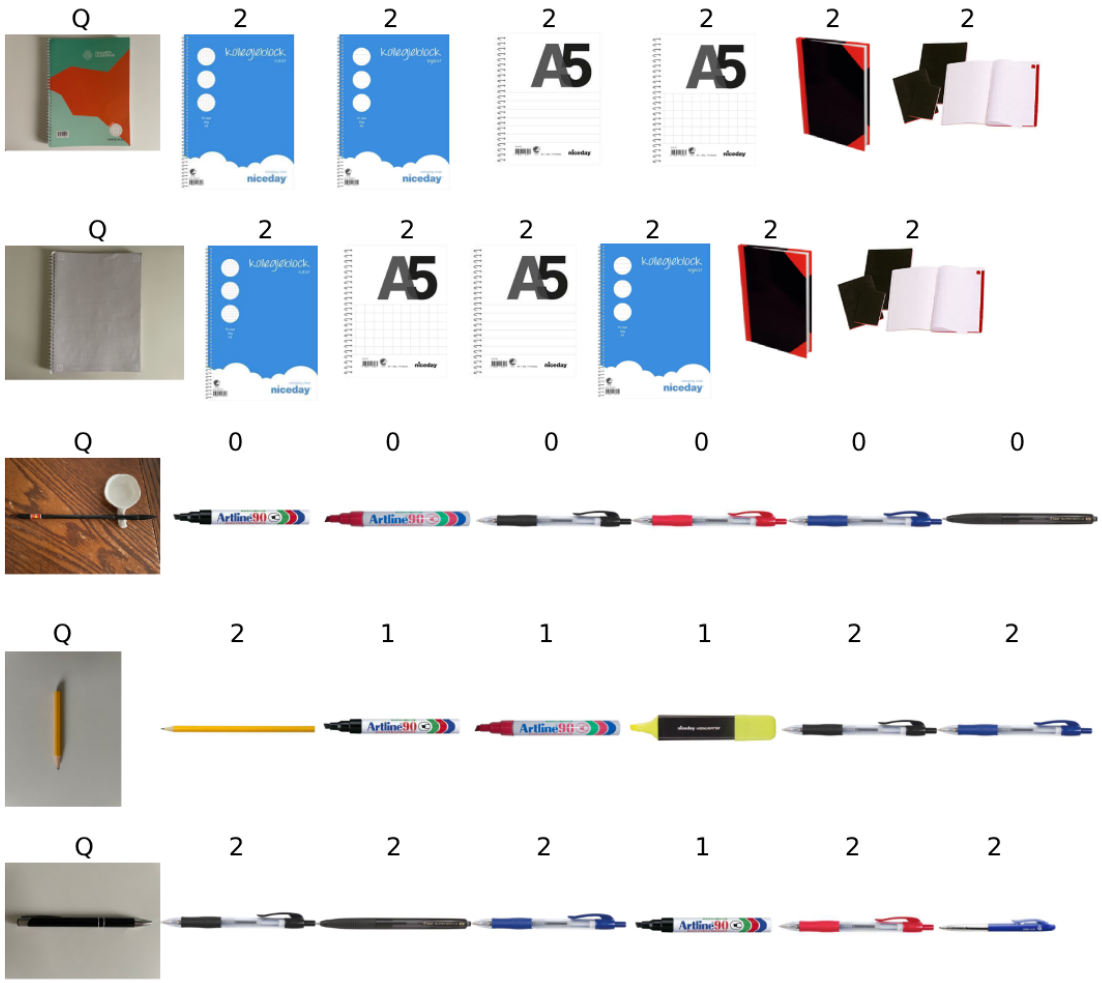


Figure 4.1: The top 6 results from five different query images. Q denotes query image, while number denotes relevance score of result.

4.3 Search Time

Properties of images used to test search time:

1. Top category: Angle grinder - File size: 128 750 bytes - Resolution: 800x538
2. Top category: Eye protection - File size: 5 227 bytes - Resolution: 259x194
3. Top category: Hammer - File size: 1 436 588 bytes - Resolution: 3264x2448
4. Top category: Helmet - File size: 4 685 471 bytes - Resolution: 3497x2564
5. Top category: Pen - File size: 6 563 114 bytes - Resolution: 4032x3024

(Resolution: width x height)

The results can be seen in table 4.7. The rows represent the vision-language model of choice, while columns represent the images. Recall that the mean of five search query times was calculated as the representative time, and that is exactly what is illustrated in the table, along with the standard error. The last column represents the mean of search times across images, where each row was calculated from preceding entries of the same row.

Time to Search	**Query Image**					
VLM	Img. 1	Img. 2	Img. 3	Img. 4	Img. 5	Avg.
Florence-2-large-hf	2.08 ± 3.22e-3	1.87 ± 5.34e-3	2.12 ± 8.88e-3	2.1 ± 11.4e-3	2.08 ± 10.9e-3	2.05
SmolVLM-256M-Instruct	5.03 ± 11.2e-3	3.62 ± 26.1e-3	0.98 ± 5.11e-3	4.78 ± 16.4e-3	2.76 ± 17.9e-3	3.43
moondream2	1.89 ± 45.2e-3	1.42 ± 4.86e-3	1.64 ± 3.17e-3	2.23 ± 7.81e-3	1.62 ± 7.27e-3	1.76
blip-image-captioning-base	0.15 ± 2.82e-3	0.12 ± 1.91e-3	0.23 ± 5.3e-3	0.22 ± 0.92e-3	0.26 ± 5.07e-3	0.2
blip2-opt-2.7b	0.48 ± 2.73e-3	0.82 ± 13.8e-3	0.64 ± 2.95e-3	0.51 ± 2.92e-3	0.81 ± 5.58e-3	0.65
git-base-coco	1.12 ± 6.78e-3	2.05 ± 5.62e-3	1.84 ± 8.04e-3	3.02 ± 10.6e-3	2.01 ± 10.3e-3	2.01

Table 4.7: The mean time it took to complete a search query for every VLM and image combination, together with the standard error. The time is in seconds.

4.4 Performance vs. Time

The trade-off between retrieval performance and search time can be seen in the plot depicted by figure 4.2. The x-axis represents search time in seconds, while the y-axis represents the nDCG@30 score. Recall that the data was collected on embedding option 1, with all-mpnet-base-v2 as the text embedding model, which goes for both the nDCG@30 data and search times. The colors of dots represent different VLMs and the legend on the right side of the figure can be referred to in order to find out which model a color corresponds to.

Figure 4.2: A plot between search time and nDCG@30 for all query images and VLMs.

4.5 Analysis & Discussion

From observing the data, it seems like the choice of vision-language model caused the biggest variance in performance, where some models gave significantly better or worse score than others. For nDCG@30, the models that generated more involved descriptions generally outperformed the models that only generated short captions. A reason for this observation could be that more detailed descriptions contained more semantic aspects which helped create a richer representation of the query image, thus being able to better match with the right products. One exception to this observation is the performance between SmolVLM-256M-Instruct and blip2-opt-2.7b, where the latter sometimes managed to outperform the former.

For the choice of embedding model, its effect on performance seems to change depending on embedding option, at least for nDCG@30. In the table 4.1 (nDCG@30, embedding option 1), we can observe that the bge models generally perform worse than the models with "all-" in their name. However, this difference largely goes away with other embedding options. As embedding option 1 generally meant short, sparse text content, this could mean that the bge models require richer text strings to perform well. Another possible explanation is that their training data simply did not suit the product dataset well, or less than all-MiniLM-L12-v2 and all-mpnet-base-v2's training data did. Overall, given that the product embeddings were rich enough, the choice of embedding model does not seem to matter significantly.

The dimension of embeddings did not seem to matter either. It would be reasonable to assume that the larger the dimension, the better the embedding is able to preserve rich semantic nuance and thus provide better performance. But the result tables are not consistent with this assumption. For example, bge-large-en-v1.5 which

produces embeddings at the highest dimension, does not generally provide better performance compared to other models. Furthermore, all-MiniLM-L12-v2's embeddings are smaller in dimension compared to bge-base-en-v1.5, yet the former does not necessarily perform worse than the latter, and in many cases provides superior performance.

An interesting observation is that option 2, where product embeddings were only created based on a VLM-generated description, provided the best performance overall on Hit@10. The most likely explanation is that since the query images and their target products' images are so similar, their generated descriptions and embeddings end up being similar as well, especially since the VLM used is the same for both products and query images. This may also explain why the variance across text embedding models is lower for option 2 relative to other options.

Figure 4.1 reveals a number of insights. First, all query images were pictures taken by the author via a smartphone, showing that the system could possibly work for people who use their own taken pictures to search, provided that nothing in the background disrupts the VLM. Second, the VLM needs to properly recognize the object in the image in order for the system to return relevant results. This is evident by the third query image, which depicted a paintbrush, yet despite there being paintbrushes in the dataset, the color markers and pens were placed at the top. This is because the VLM (Florence-2-large-hf) described the object as a pen rather than a paintbrush, which will better match with the ballpoint and color marker pens rather than the target items. Note that the pencil image only returned one pencil because there was only one of them in the dataset.

As for time required to search, it is clear that the choice of vision-language model significantly affects how long it takes. For example, none of blip-image-captioning-base's mean times ever reached half a second, while SmolVLM-256M-Instruct would require around five seconds to perform its search on image 1. This phenomenon was evident during development and evaluation, as the choice of model would noticeably alter the time it took for a description to be generated.

In fact, the primary factor explaining the time differences between VLMs seems to be the model's output length. VLMs that generated longer descriptions generally took longer time than models whose output was a mere short caption. Note how SmolVLM-256M-Instruct's times across query images vary a lot relative to other models and this is precisely due to how the model's output length tended to vary across images. Sometimes it would produce a short caption, while on other images it could generate multiple longer paragraphs. In contrast, other models tended to generate captions at similar lengths regardless of the image, explaining why they do not show as much variance across queries. A trade-off seems to emerge where longer descriptions provide better retrieval performance, but at the cost of extra search time.

But does that trade-off really exist? The scatter plot in figure 4.2 provides a better overview of the search time for each model, along with their nDCG@30 performance. Most of the points for any given model are clustered over the top-half, and these represent query images for which the system got a high nDCG@30 score on. On

some models like blip-image-captioning-base and git-base-coco, more points can be seen on the lower half, meaning that their retrieval performance was relatively worse. The wider a model's point distribution is, the higher variance it has in search time, with the most prominent example being SmolVLM-256M-Instruct which has points spanning from sub-1 second, up to over 7 seconds. Overall, the plot shows that longer search time does not necessarily facilitate better retrieval performance, and that developers should experiment with different VLMs to see if they can achieve adequate performance on both aspects before having to trade one for the other.

Another interesting question is whether file size or image resolution actually affect the time taken to perform a search query. An intuitive assumption would be that the bigger the file size and the higher the resolution, the longer time it would take. But this does not seem to be the case as observed in the table, where e.g., the 5th image is clearly the heaviest overall, yet only for blip-image-captioning-base did it actually take the longest. Meanwhile, image 2 is the lightest by far, but would sometimes take longer than even the heaviest image depending on the VLM. Part of this is likely because most models will resize the image in order to fit the architecture due to the input having to match the dimensions of the model, which would reduce the impact of resolution. On the other hand, file size likely does not matter as it may be more reflective of image compression results rather than anything that affects VLM processing speed. In other words, a bigger image might take less time to process than a smaller image because the latter was able to be compressed further than the former, while attributes that actually affect processing speed are heavier for the smaller image.

As lightly noted before, the time evaluation was tested on the equipment mentioned in chapter 3, and this is another significant variable. Most models were ran on the listed GPU, and since computation time is highly dependent on the quality of the GPU, this implies that the search system would be infeasible to use with a significantly worse GPU, or on a CPU alone. In practice, if this were to be implemented in a commercial or real-world context, any interactions with the deep learning models would likely be handled by a server equipped with the computational resources to deal with search queries, hosted from the developer's side.

Recall that the minimum acceptable outcome was defined as the system being able to retrieve at least three relevant entries in the top 10 results, for three different categories (in terms of retrieval performance) while taking at most a minute to execute a search query. Most of the VLM, text embedding model and embedding option combinations managed to not only satisfy this outcome, but exceed it. Examples of specific search results can be found in the appendix, chapter B.

The two research questions should be addressed as well, and in short, the answer to both is yes. The first one entailed fulfilling the minimum acceptable outcome while using open source models without alteration (e.g. performing fine-tuning), which the system did. The second question simply asked if the search queries managed to require less than five seconds to complete, and this was fulfilled by all VLMs except for SmolVLM-256M-Instruct with image 1.

The discussion would not be complete without asking whether the visual search

system would be viable in a real-world context. Overall, a firm conclusion cannot be drawn since the system was never implemented in such an environment, but if we isolate the question to just the retrieval performance, the answer should be yes, as long as the VLM of choice is able to correctly recognize the item in the query image, and describe it in enough detail. As for time taken to search, the evaluation data demonstrates that execution times are reasonably fast in individual circumstances, but whether this would scale to a situation where hundreds, if not thousands of users are all trying to search at the same time, has yet to be seen.

5

Conclusion

5.1 Summary

In summary, a new method for implementing visual search was explored, where image descriptions from a VLM were used as the basis for a similarity search. It was found to be viable in terms of retrieving relevant products, with the best models providing nDCG@30 scores in the mid-80 percentage range (the best combination gave 85.9%). Satisfactory search times were also achieved, although this was on an individual level. Future work is required to find out whether it would work in a true production-ready context, if it would scale to both increasing amounts of users and product database, and how its performance compares to other methods. But it is a potentially promising option for developers who do not have the resources to train or fine-tune a model while also being easy to implement.

5.2 Limitations

Naturally, the project has limitations that are important to highlight. The performance of an information retrieval system is highly dependent on the data that it has been tasked to retrieve. It was clear during development that no matter what evaluation metric was chosen, one could alter the dataset, query set or product categorization in order to earn a perfect score. Therefore, a limitation is that the evaluation data from this thesis cannot be directly compared to results of other work since the datasets used in each of them are different from each other and to this project.

The lack of feedback from the target audience is another limitation. As mentioned before, the evaluation should also ideally take into account target audience feedback, which could not only give insights into system adjustments, but also how the data should be structured and classified in the first place. Even though the nDCG@30 scores demonstrated that the system was capable of retrieving relevant entries, it does not necessarily mean that the target audience would find utility in the tool. Note that in this case, the target audience would be customers of the e-commerce website that the product data was scraped from.

Categorization of products itself is a possible limitation. For example, as is evident in figure 4.1, pencils, ballpoint pens and color markers were all in the same top category, while paintbrushes were in a completely different one. Some may argue

that all of them should be under the same top category, others may make the point that they should all be separated. This is especially important to consider if the developer wants to add an image/object classification technique and narrow down the searching based on class. Either way, the most optimal categorization likely depends on how the target audience uses the system. If the users are buying color markers after using paintbrushes as query, it would make sense to categorize them together on some level, but if this is not happening, then separating them would be more appropriate.

As the dataset only consisted of 203 products, this means that the thesis' conclusions are mostly only suitable for e-commerce databases of this scale. Different indexing methods may have to be explored in order to scale the method to larger databases without sacrificing search query time. It is also worth mentioning that the subset of query images used to evaluate exact matches, being only 11, might have been too small and thus subjected the Hit@10 results to chance.

All text embedding models used in this thesis were trained on the English language, which was sufficient given that almost all metadata associated with the products were in English, in addition to the vision-language models producing their captions or descriptions in English. But if the product data were to be in another language instead, some extra steps would have to be taken. The text embedding models would have to be changed in order to suit the target language, either by replacing the English model with another one trained in that language, or with a multilingual model. The former option also requires the VLM produce captions in the target language. Either way, the upshot is that the model's performance on other languages are yet to be seen, providing grounds for future work.

5.3 Future Work

In the present system, the product embeddings are created from one of three means: by concatenating the name and description, from a generated product description, or a combination of both. For future work, this could be expanded. Certain VLMs (including some used in this thesis) allow for more complicated text outputs, or the ability to be prompted. This could be utilized to create more fields for each product, like color, style, size, and other attributes, before converting all fields into one embedding.

When evaluating the system, whenever a given VLM was used to generate product descriptions, it would also be used to generate a description of the query image. But, unlike the text embedding model, the VLM does not have to be the same for both. Another method would be to use a larger VLM to generate detailed product descriptions, and a lighter VLM for querying. This could potentially make target products easier to match with, but also reduce the search time. Overall, the "best" VLM for product/entry descriptions may not necessarily be the same as the "best" VLM for querying, and further work could explore multiple combinations.

In this thesis, many of the available parameters were either set as default or as a value recommended by the model developers. If any future work make use of the same

models, there may be potential for improvement by experimenting with different parameter values. In addition, some VLMs have the ability to be prompted, allowing for different descriptions. Not only could this be useful for product descriptions, but also for optimization of query descriptions. SmolVLM-256M-Instruct in particular could likely be better than the data depicts, as not only does it allow for custom prompts that may facilitate better retrieval performance, its output length was not limited in this thesis, which meant longer descriptions and processing times than what was likely optimal.

One of the limitations was that evaluation data could not be directly compared to other related work and their methods due to different datasets. Therefore, a possibility for future work is to implement both this thesis's method and another method, and run search experiments on the same dataset using any of the standard evaluation metrics (nDCG@K, mAP@K, etc.) This would provide a direct comparison between methods and highlight which method is (contextually) superior.

Bibliography

[1] M. S. Lew, N. Sebe, C. Djeraba, and R. Jain, "Content-based multimedia information retrieval: State of the art and challenges", *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, vol. 2, no. 1, pp. 1–19, 2006.

[2] S. Thakur, *New ways for marketers to reach customers with AI Overviews and Lens*, `https://blog.google/products/ads-commerce/google-lens-ai-overviews-ads-marketers/`, 2024. Accessed: Dec. 7, 2025.

[3] M. Mahadevan, *6 new visual search features that help you quickly find what you're looking for on amazon*, `https://www.aboutamazon.com/news/retail/visual-search-shopping-features`, 2024. Accessed: Dec. 7, 2025.

[4] M. Flickner et al., "Query by image and video content: The QBIC system", *computer*, vol. 28, no. 9, pp. 23–32, 1995.

[5] J. R. Bach et al., "Virage image search engine: An open framework for image management", in *Storage and retrieval for still image and video databases IV*, SPIE, vol. 2670, 1996, pp. 76–87.

[6] H. Hu et al., "Web-scale responsive visual search at bing", in *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, 2018, pp. 359–367.

[7] Y. Jing et al., "Visual search at pinterest", in *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, 2015, pp. 1889–1898.

[8] F. Yang et al., "Visual search at ebay", in *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, 2017, pp. 2101–2110.

[9] H. Al-Lohibi, T. Alkhamisi, M. Assagran, A. Aljohani, and A. O. Aljahdali, "Awjedni: A reverse-image-search application", *ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal*, vol. 9, no. 3, p. 49, 2020.

[10] F. H. Araujo et al., "Reverse image search for scientific data within and beyond the visible spectrum", *Expert Systems with Applications*, vol. 109, pp. 35–48, 2018.

[11] R. Shiau et al., "Shop the look: Building a large scale visual shopping system at pinterest", in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020, pp. 3203–3212.

[12] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks", in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 1492–1500.

[13] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition", *CoRR*, vol. abs/1512.03385, 2015. arXiv: 1512.03385. [Online]. Available: http://arxiv.org/abs/1512.03385.

[14] Y. Yada, S. Akiyama, R. Watanabe, Y. Ueno, Y. Shido, and A. Rusli, *Improving visual recommendation on e-commerce platforms using vision-language models*, 2025. arXiv: 2510.13359 [cs.IR]. [Online]. Available: https://arxiv.org/abs/2510.13359.

[15] X. Zhai, B. Mustafa, A. Kolesnikov, and L. Beyer, *Sigmoid loss for language image pre-training*, 2023. arXiv: 2303.15343 [cs.CV]. [Online]. Available: https://arxiv.org/abs/2303.15343.

[16] S. Ren, K. He, R. B. Girshick, and J. Sun, "Faster R-CNN: towards real-time object detection with region proposal networks", *CoRR*, vol. abs/1506.01497, 2015. arXiv: 1506.01497. [Online]. Available: http://arxiv.org/abs/1506.01497.

[17] dvgodoy, *Transformer, full architecture.png*, https://commons.wikimedia.org/wiki/File:Transformer,_full_architecture.png, 2024. Accessed: Jan. 25, 2026.

[18] A. Vaswani et al., "Attention is all you need", *Advances in neural information processing systems*, vol. 30, 2017.

[19] H.-J. Hwang, S.-G. Jeong, and W.-J. Hwang, "Ultrawideband non-line-of-sight classification using transformer-convolutional neural networks", *IEEE Access*, 2025.

[20] J. Wang et al., *Git: A generative image-to-text transformer for vision and language*, 2022. arXiv: 2205.14100 [cs.CV]. [Online]. Available: https://arxiv.org/abs/2205.14100.

[21] B. Xiao et al., "Florence-2: Advancing a unified representation for a variety of vision tasks", in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024, pp. 4818–4829.

[22] M. Ding, B. Xiao, N. Codella, P. Luo, J. Wang, and L. Yuan, "Davit: Dual attention vision transformers", in *European conference on computer vision*, Springer, 2022, pp. 74–92.

[23] J. Wang et al., "Milvus: A purpose-built vector data management system", in *Proceedings of the 2021 International Conference on Management of Data*, ser. SIGMOD '21, Virtual Event, China: Association for Computing Machinery, 2021, pp. 2614–2627, ISBN: 9781450383431. DOI: 10.1145/3448016.3457550. [Online]. Available: https://doi.org/10.1145/3448016.3457550.

[24] A. Marafioti et al., "Smolvlm: Redefining small and efficient multimodal models", *arXiv preprint arXiv:2504.05299*, 2025.

[25] vikhyatk, *Moondream 2*, https://huggingface.co/vikhyatk/moondream2.

[26] J. Li, D. Li, C. Xiong, and S. Hoi, "Blip: Bootstrapping language-image pre-training for unified vision-language understanding and generation", in *International conference on machine learning*, PMLR, 2022, pp. 12 888–12 900.

[27] J. Li, D. Li, S. Savarese, and S. Hoi, "Blip-2: Bootstrapping language-image pre-training with frozen image encoders and large language models", in *International conference on machine learning*, PMLR, 2023, pp. 19 730–19 742.

[28] *all-MiniLM-L12-v2*, https://huggingface.co/sentence-transformers/all-MiniLM-L12-v2.

[29] *all-mpnet-base-v2*, https://huggingface.co/sentence-transformers/all-mpnet-base-v2.

[30] BAAI, *bge-base-en-v1.5*, https://huggingface.co/BAAI/bge-base-en-v1.5.

[31] S. Xiao, Z. Liu, P. Zhang, and N. Muennighoff, *C-pack: Packaged resources to advance general chinese embedding*, 2023. arXiv: 2309.07597 [cs.CL].

A

Appendix 1

A.1 Model Parameters

Each VLM and their processor were provided by Hugging Face's Transformers module in Python. The processor and model's generate function sometimes require parameters and as such a list of all noteworthy parameters that were used in this thesis can be found below. Sometimes the model's initialization had important parameters and in such cases, they can be found below as well. Parameters not explicitly listed were not manually set and consequently remained their default values. Note that no parameters were set for any text embedding models. Therefore the list only concerns VLMs.

Florence-2-large-hf

Processor:

- prompt = "<MORE_DETAILED_CAPTION>"

Generation:

- max_new_tokens = 4096
- num_beams = 3
- do_sample = False

SmolVLM-256M-Instruct

Processor:

- message = [{ "role": "user", "content": [{"type": "image"}, {"type": "text", "text": "Can you describe this image?"}] },]
- add_generation_prompt = True

Generation:

- max_new_tokens = 500

Moondream 2

Initialization:

- revision="2025-06-21"

Generation:

- prompt = “Normal”

blip-image-captioning-base

None

blip2-opt-2.7b

Model Initialization:

- load_in_8bit = True

Note that this was required given the hardware equipment. A better GPU could probably skip this parameter.

git-base-coco

Generation:

- max_length = 50

B

Raw Search Data

The following chapter contains raw search data from some combinations of vision-language model, text embedding model and embedding option (listing all data would be excessive).

Each evaluation entry begins with its image file name, followed by the query's top and subcategory. The numbers in brackets represent the relevance scores of search results, ordered in the same way the actual results were ranked. At the end of the row, the evaluation score is presented.

Florence-2-large-hf - all-mpnet-base-v2 - Option 1

query_safetyvisor1.jpg (eyeprotection, subcategory: 195): [0, 0, 0, 2, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 2, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0] - 0.48196175273168834

query_safetyvisor2.png (eyeprotection, subcategory: 195): [0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 2, 2, 0, 1, 1, 0, 0, 0] - 0.39741777156325925

query_safetyvisor3.jpg (eyeprotection, subcategory: 195): [0, 0, 0, 0, 2, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.36386565134798254

queryc0_eyeprotection3.jpg (eyeprotection, subcategory: 195): [0, 0, 0, 2, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 2, 2, 0, 1, 0, 0, 0, 0, 1, 1, 1] - 0.46846010954056905

queryc0_eyeprotection2.jpeg (eyeprotection, subcategory: 193): [2, 2, 1, 1, 1, 2, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0] - 0.8853512044636005

query_safetygoggles1.jpg (eyeprotection, subcategory: 193): [2, 2, 2, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0] - 0.9809465588025958

queryc0_eyeprotection1.jpeg (eyeprotection, subcategory: 193): [2, 2, 2, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.9043369487435851

query_safetyglasses2.jpg (eyeprotection, subcategory: 194): [2, 2, 2, 2, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.9903129469616859

queryc0_eyeprotection4.jpg (eyeprotection, subcategory: 194): [2, 2, 2, 1, 2, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0] - 0.9212829555334595

query_safetyglasses1.jpg (eyeprotection, subcategory: 194): [2, 2, 2, 2, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.9643917934107323

queryc0_anglegrinder3.jpeg (anglegrinder, subcategory: 283): [2, 0, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.9060254355346824

queryM_anglegrinder1.jpg (anglegrinder, subcategory: 283): [2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 1.0

queryc0_anglegrinder1.png (anglegrinder, subcategory: 283): [2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 1.0

queryc0_anglegrinder2.jpg (anglegrinder, subcategory: 283): [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0] - 0.34183456755229635

queryc0_helmet2.jpg (helmet, subcategory: 198): [2, 2, 2, 2, 2, 2, 2, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.9963488021549354

queryc0_helmet3.jpg (helmet, subcategory: 198): [2, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.5452079709581737

queryM_kaskhelmetwhite.jpg (helmet, subcategory: 198): [2, 2, 2, 2, 2, 2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 1.0

queryc0_helmet1.png (helmet, subcategory: 198): [2, 2, 2, 2, 0, 0, 0, 0, 0, 0, 2, 0, 2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.9129098800404627

queryc0p_hammer4.jpg (hammer, subcategory: 81): [2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 1.0

queryc0_hammer1.jpg (hammer, subcategory: 81): [2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 1.0

queryc0p_hammer2.jpg (hammer, subcategory: 81): [2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 1.0

queryc0_hammer3.jpg (hammer, subcategory: 81): [2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 1.0

queryc0p_hammer1.jpg (hammer, subcategory: 81): [2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 1.0

queryc0p_hammer3.jpg (hammer, subcategory: 81): [2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 1.0

queryc0_hammer2.jpeg (hammer, subcategory: 81): [2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 1.0

queryM_trousersmodaflame2.jpg (trousers, subcategory: 247): [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 2, 2, 2, 0, 2, 0, 0, 1, 1, 1, 1, 1] - 0.2977723830634845

queryM_trousersmodaflame1.jpg (trousers, subcategory: 247): [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2] - 0.7172422675672576

query_flameprooftrousers1.jpg (trousers, subcategory: 247): [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0] - 0.5539778780832871

queryc0_trousers1.jpg (trousers, subcategory: 246): [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 1, 0, 0] - 0.9762568639418341

queryM_trousers1.jpg (trousers, subcategory: 246): [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 0, 1, 0] - 0.976177478093134

queryc0_trousers3.jpg (trousers, subcategory: 246): [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1] - 1.0

queryM_trousers2.jpg (trousers, subcategory: 246): [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 0, 0, 0] - 0.9682924219799222

queryc0_trousers2.jpeg (trousers, subcategory: 246): [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1] - 1.0

queryM_trousers3.jpg (trousers, subcategory: 246): [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 0, 0, 0] - 0.9682924219799222

query_maskingtape1.jpg (tape, subcategory: 51): [1, 1, 1, 1, 1, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.7923238755368758

query_maskingtape2.jpg (tape, subcategory: 51): [1, 1, 1, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.8234462437389305

queryc0_tape1.png (tape, subcategory: 51): [1, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0] - 0.8913281993453693

query_electrictape3.jpeg (tape, subcategory: 52): [2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 1.0

query_electrictape2.jpg (tape, subcategory: 52): [1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.7840265357070773

query_electrictape1.jpg (tape, subcategory: 52): [2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 1.0

query_garbagetape3.jpg (tape, subcategory: 50): [1, 1, 1, 2, 1, 1, 1, 1, 2, 2, 2, 1, 2, 1, 1, 2, 1, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.7697119024802567

query_garbagetape4.jpg (tape, subcategory: 50): [1, 1, 1, 1, 1, 1, 1, 2, 2, 1, 2, 2, 2, 1, 1, 1, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.749667992264667

query_garbagetape1.jpg (tape, subcategory: 50): [1, 2, 1, 1, 1, 1, 1, 2, 2, 2, 2, 1, 1, 1, 1, 2, 2, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.8068726180258535

query_garbagetape2.jpg (tape, subcategory: 50): [1, 1, 1, 2, 1, 1, 1, 2, 2, 1, 2, 2, 2, 2, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.7801810699488456

query_packingtape1.jpg (tape, subcategory: 53): [1, 1, 1, 1, 2, 1, 2, 1, 1, 1, 1, 1, 1, 2, 1, 2, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.7708882341485257

queryc0_tape2.jpeg (tape, subcategory: 53): [1, 1, 1, 1, 1, 1, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 2, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.7655647748472161

query_selfamalgamatingtape1.jpg (tape, subcategory: 53): [1, 1, 1, 2, 1, 1, 2, 2, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.7894589867747565

query_reflectiontape1.jpeg (tape, subcategory: 53): [1, 1, 1, 2, 2, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.7898701910128659

queryc0p_tape2.jpg (tape, subcategory: 54): [2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 1.0

queryc0_tape3.jpg (tape, subcategory: 54): [1, 1, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.8577166423015301

queryc0p_tape1.jpg (tape, subcategory: 54): [2, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.9376399802667761

queryM_modaflamejacketorange1.jpg (jacket, subcategory: 238): [1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 1, 2, 2, 1, 1, 2] - 0.6780075498401228

queryc0_winterjacket2.jpeg (jacket, subcategory: 238): [0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.08070694611925983

query_flameproofjacket1.jpg (jacket, subcategory: 238): [0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0] - 0.13885088850854058

query_genericwinterjacket2.jpg (jacket, subcategory: 239): [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 2, 2, 2, 0, 1, 1, 1, 0, 2, 0, 2, 2, 0, 0, 0] - 0.3347323879070563

queryc0_winterjacket3.jpeg (jacket, subcategory: 239): [0, 2, 2, 2, 2, 0, 2, 0, 2, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.5222968352516961

query_genericwinterjacket1.jpg (jacket, subcategory: 239): [2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1] - 1.0

query_genericwinterjacket3.jpg (jacket, subcategory: 239): [2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1] - 1.0

query_parkasjacket3.png (jacket, subcategory: 240): [1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0] - 0.6959440898196334

query_parkasjacket1.png (jacket, subcategory: 240): [1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0] - 0.7597976402184587

queryc0_winterjacket1.jpeg (jacket, subcategory: 240): [2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.8510560034645807

query_parkasjacket2.jpg (jacket, subcategory: 240): [2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1] - 1.0

query_swissknife2.jpg (knife, subcategory: 85): [1, 1, 1, 2, 1, 2, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.7901136970493384

query_swissknife1.jpg (knife, subcategory: 85): [2, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.9183857015702735

query_pocketknife1.jpeg (knife, subcategory: 85): [2, 1, 1, 1, 1, 1, 1, 1, 2, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.9163702783327445

query_pocketknife2.jpg (knife, subcategory: 85): [2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.912921559666639

queryc0p_knife1.jpg (knife, subcategory: 86): [1, 2, 1, 1, 2, 2, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.8333531792146044

queryc0p_knife2.jpg (knife, subcategory: 86): [2, 1, 1, 1, 2, 1, 2, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.9095712781021493

queryc0_knife2.jpg (knife, subcategory: 86): [2, 1, 1, 1, 2, 1, 2, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.9095712781021493

queryc0p_knife3.jpg (knife, subcategory: 86): [1, 2, 1, 1, 2, 1, 2, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.8283173045321042

query_snapoffknife2.jpg (knife, subcategory: 83): [1, 2, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 2, 1, 0, 1, 0, 0, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0] - 0.7070644828530265

query_snapoffknife1.jpg (knife, subcategory: 83): [1, 1, 1, 1, 1, 2, 2, 1, 1, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.7377648502044173

queryc0_knife4.jpg (knife, subcategory: 83): [1, 1, 1, 1, 2, 1, 1, 2, 1, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.7403342308267965

queryc0_knife1.jpeg (knife, subcategory: 84): [2, 2, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1] - 0.9438279399801331

queryc0_knife3.jpeg (knife, subcategory: 84): [1, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.8763043978155691

queryM_bahcoknife1.jpg (knife, subcategory: 84): [1, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.8763043978155691

queryc0_paintbrush2.jpg (paintbrush, subcategory: 172): [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 1.0

queryc0p_paintbrush2.jpg (paintbrush, subcategory: 172): [0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 2, 2, 0, 2, 2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 2, 0, 0] - 0.3185288213993854

queryc0_paintbrush3.jpg (paintbrush, subcategory: 172): [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 1.0

queryc0p_paintbrush1.jpg (paintbrush, subcategory: 172): [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 2, 2, 0, 2, 0, 0, 0, 0, 0, 0] - 0.4161326626939686

queryc0_paintbrush1.jpg (paintbrush, subcategory: 172): [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 1.0

queryc0_winterboots2.jpg (boots, subcategory: 251): [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2] - 1.0

queryc0_winterboots3.jpg (boots, subcategory: 251): [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2] - 1.0

queryc0_winterboots1.jpg (boots, subcategory: 251): [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2] - 1.0

queryc0_pen1.jpeg (pen, subcategory: 210): [2, 2, 2, 2, 2, 1, 1, 1, 1, 2, 0, 0, 0, 0, 0, 0, 0, 1, 1, 2, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0] - 0.9402382739582815

queryc0_pen2.jpg (pen, subcategory: 210): [2, 2, 1, 1, 2, 2, 1, 2, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 2, 0, 1, 2, 0, 0, 0, 0, 0, 0, 0, 0] - 0.9007042522997772

queryc0p_pen1.jpg (pen, subcategory: 210): [2, 1, 1, 1, 2, 2, 1, 2, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 1, 2] - 0.7761550901174052

queryc0p_pen2.jpg (pen, subcategory: 210): [2, 2, 2, 1, 2, 2, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 2, 1, 1, 0, 0, 2, 0, 1, 1, 0, 0, 0, 0, 0] - 0.8902618535499087

queryM_nicedayhighlighterblue1.jpg (pen, subcategory: 209): [1, 1, 1, 1, 2, 2, 1, 2, 2, 2, 2, 2, 0, 1, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0] - 0.732607267454723

queryM_forayhighlighters1.jpg (pen, subcategory: 209): [2, 2, 2, 1, 1, 1, 2, 1, 2, 2, 1, 2, 2, 2, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.9430842207462352

queryc0_pen3.jpg (pen, subcategory: 209): [1, 2, 1, 2, 2, 2, 1, 1, 1, 2, 2, 1, 2, 1, 2, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.8433896922442954

queryc0_notebook1.jpg (notebook, subcategory: 206): [2, 2, 2, 2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 1.0

queryc0_notebook3.jpg (notebook, subcategory: 206): [2, 2, 2, 2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 1.0

queryc0p_notebook2.jpg (notebook, subcategory: 206): [2, 2, 2, 2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 1.0

queryc0p_notebook1.jpg (notebook, subcategory: 206): [2, 2, 2, 2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 1.0

queryc0_notebook2.jpg (notebook, subcategory: 206): [2, 2, 2, 2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 1.0

Mean NDCG@30: 0.828646482938614

Florence-2-large-hf - bge-large-en-v1.5 - Option 2

query_snapoffknife1.jpg (knife, subcategory: 83): [1, 1, 1, 2, 1, 1, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 2, 1] - 0.6054797845799518

query_snapoffknife2.jpg (knife, subcategory: 83): [0, 0, 1, 0, 2, 1, 1, 0, 0, 2, 0, 1, 0, 0, 0, 0, 0, 2, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0] - 0.48850978991706695

queryc0_knife4.jpg (knife, subcategory: 83): [2, 2, 1, 1, 2, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.8625950512252275

query_swissknife2.jpg (knife, subcategory: 85): [2, 1, 2, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.810815469440969

query_pocketknife2.jpg (knife, subcategory: 85): [1, 1, 1, 2, 1, 1, 1, 2, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0] - 0.7642214484072242

query_swissknife1.jpg (knife, subcategory: 85): [2, 2, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.9627639134107672

query_pocketknife1.jpeg (knife, subcategory: 85): [2, 1, 1, 1, 1, 0, 1, 1, 1, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.8368603457661018

queryc0p_knife2.jpg (knife, subcategory: 86): [2, 2, 1, 2, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.9540320124718654

queryc0_knife2.jpg (knife, subcategory: 86): [2, 2, 1, 1, 2, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.9389621743721918

queryc0p_knife3.jpg (knife, subcategory: 86): [2, 2, 2, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0] - 0.9534013565120143

queryc0p_knife1.jpg (knife, subcategory: 86): [2, 1, 2, 2, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.8860822869216746

queryc0_knife3.jpeg (knife, subcategory: 84): [1, 2, 2, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.8405483500538402

queryM_bahcoknife1.jpg (knife, subcategory: 84): [2, 1, 2, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.9318530828145098

queryc0_knife1.jpeg (knife, subcategory: 84): [0, 1, 2, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 2] - 0.5580906101272112

queryc0p_notebook1.jpg (notebook, subcategory: 206): [2, 2, 2, 2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 1.0

queryc0_notebook3.jpg (notebook, subcategory: 206): [2, 2, 2, 2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 1.0

queryc0p_notebook2.jpg (notebook, subcategory: 206): [2, 2, 2, 2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 1.0

queryc0_notebook1.jpg (notebook, subcategory: 206): [2, 2, 2, 0, 0, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.9234967178511228

queryc0_notebook2.jpg (notebook, subcategory: 206): [2, 2, 2, 2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 1.0

queryc0_paintbrush1.jpg (paintbrush, subcategory: 172): [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 1.0

queryc0_paintbrush2.jpg (paintbrush, subcategory: 172): [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 1.0

queryc0p_paintbrush1.jpg (paintbrush, subcategory: 172): [0, 0, 0, 0, 0, 0, 2, 2, 2, 2, 2, 0, 2, 2, 0, 2, 2, 0, 0, 0, 0, 0, 2, 2, 2, 2, 0, 0, 0, 0] - 0.47972675266044484

queryc0_paintbrush3.jpg (paintbrush, subcategory: 172): [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 1.0

queryc0p_paintbrush2.jpg (paintbrush, subcategory: 172): [0, 0, 0, 2, 2, 0, 0, 0, 0, 0, 0, 0, 2, 2, 0, 0, 0, 2, 2, 2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 2] - 0.3803479917769653

query_genericwinterjacket2.jpg (jacket, subcategory: 239): [1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0] - 0.7442222429394846

queryc0_winterjacket3.jpeg (jacket, subcategory: 239): [0, 2, 2, 2, 2, 2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1] - 0.6554305074412633

query_genericwinterjacket3.jpg (jacket, subcategory: 239): [2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1] - 1.0

query_genericwinterjacket1.jpg (jacket, subcategory: 239): [2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1] - 1.0

queryM_modaflamejacketorange1.jpg (jacket, subcategory: 238): [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1] - 0.8801462630153993

query_flameproofjacket1.jpg (jacket, subcategory: 238): [2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 0, 0, 0, 0, 0, 0, 2, 2, 2, 2, 2, 2] - 0.7458799846995616

queryc0_winterjacket2.jpeg (jacket, subcategory: 238): [0, 0, 0, 0, 0, 0, 0, 0, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 2, 2, 2, 2, 2, 2] - 0.3605959346778621

query_parkasjacket1.png (jacket, subcategory: 240): [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1] - 0.7696789337695739

query_parkasjacket2.jpg (jacket, subcategory: 240): [1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0] - 0.7288840724962289

queryc0_winterjacket1.jpeg (jacket, subcategory: 240): [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1] - 0.7696789337695739

query_parkasjacket3.png (jacket, subcategory: 240): [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1] - 0.7531156243909263

queryc0_anglegrinder3.jpeg (anglegrinder, subcategory: 283): [2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 1.0

queryc0_anglegrinder1.png (anglegrinder, subcategory: 283): [2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 1.0

queryM_anglegrinder1.jpg (anglegrinder, subcategory: 283): [2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 1.0

queryc0_anglegrinder2.jpg (anglegrinder, subcategory: 283): [0, 0, 0, 2, 0, 2, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.5049197748991977

queryM_kaskhelmetwhite.jpg (helmet, subcategory: 198): [2, 2, 2, 2, 0, 2, 0, 0, 2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.9578563942987676

queryc0_helmet3.jpg (helmet, subcategory: 198): [2, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 2, 2, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.5257586652867632

queryc0_helmet1.png (helmet, subcategory: 198): [2, 2, 2, 2, 2, 2, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0] - 0.9716024758996962

queryc0_helmet2.jpg (helmet, subcategory: 198): [2, 2, 2, 2, 2, 2, 2, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.9907620440285202

queryc0_tape3.jpg (tape, subcategory: 54): [1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 2, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0] - 0.7803678318394422

queryc0p_tape2.jpg (tape, subcategory: 54): [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 2, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.7484148205904851

queryc0p_tape1.jpg (tape, subcategory: 54): [2, 1, 2, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1] - 0.8953598163339129

query_packingtape1.jpg (tape, subcategory: 53): [1, 1, 1, 2, 2, 1, 1, 1, 2, 1, 1, 1, 2, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.7781862781453902

query_selfamalgamatingtape1.jpg (tape, subcategory: 53): [1, 1, 1, 1, 1, 2, 2, 1, 1, 1, 1, 2, 1, 1, 2, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.7488930645691815

queryc0_tape2.jpeg (tape, subcategory: 53): [1, 1, 1, 1, 1, 2, 1, 1, 1, 2, 1, 2, 2, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.7622211819942865

query_reflectiontape1.jpeg (tape, subcategory: 53): [2, 2, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0] - 0.9221991147338143

query_electrictape1.jpg (tape, subcategory: 52): [2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.9045641624499634

query_electrictape2.jpg (tape, subcategory: 52): [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 2, 0, 0, 2, 0, 0, 0, 0, 0, 0, 1, 0, 0] - 0.7564861279736408

query_electrictape3.jpeg (tape, subcategory: 52): [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.6685697749757668

query_garbagetape3.jpg (tape, subcategory: 50): [1, 2, 2, 2, 2, 1, 1, 2, 2, 1, 1, 1, 1, 1, 0, 1, 2, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0] - 0.8760166474552881

query_garbagetape4.jpg (tape, subcategory: 50): [2, 2, 2, 1, 2, 2, 1, 1, 1, 2, 1, 1, 1, 2, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.9514056186693318

query_garbagetape1.jpg (tape, subcategory: 50): [2, 1, 2, 2, 2, 2, 1, 1, 2, 1, 1, 1, 1, 1, 2, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.9224940229119308

query_garbagetape2.jpg (tape, subcategory: 50): [2, 1, 2, 1, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 2, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.9114041738475696

queryc0_tape1.png (tape, subcategory: 51): [1, 2, 2, 1, 2, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.8327358541227762

query_maskingtape2.jpg (tape, subcategory: 51): [2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.9782894781001141

query_maskingtape1.jpg (tape, subcategory: 51): [2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0] - 0.9979099240243131

queryc0_winterboots3.jpg (boots, subcategory: 251): [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2] - 1.0

queryc0_winterboots2.jpg (boots, subcategory: 251): [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2] - 1.0

queryc0_winterboots1.jpg (boots, subcategory: 251): [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2] - 1.0

queryc0_trousers1.jpg (trousers, subcategory: 246): [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2] - 0.9737668020215481

queryM_trousers2.jpg (trousers, subcategory: 246): [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2] - 0.9737668020215481

queryM_trousers3.jpg (trousers, subcategory: 246): [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1] - 1.0

queryc0_trousers2.jpeg (trousers, subcategory: 246): [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1] - 1.0

queryM_trousers1.jpg (trousers, subcategory: 246): [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2] - 0.9737668020215481

queryc0_trousers3.jpg (trousers, subcategory: 246): [2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2] - 0.9459616455050176

query_flameprooftrousers1.jpg (trousers, subcategory: 247): [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 0, 0, 0, 0, 0, 0, 1, 1, 1] - 0.6768955069846256

queryM_trousersmodaflame2.jpg (trousers, subcategory: 247): [0, 0, 0, 0, 0, 0, 0, 0, 2, 2, 2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.27929585101420185

queryM_trousersmodaflame1.jpg (trousers, subcategory: 247): [2, 2, 2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.6841708011779141

queryc0p_hammer1.jpg (hammer, subcategory: 81): [2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 1.0

queryc0p_hammer2.jpg (hammer, subcategory: 81): [2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 1.0

queryc0p_hammer3.jpg (hammer, subcategory: 81): [2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 1.0

queryc0_hammer2.jpeg (hammer, subcategory: 81): [2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 1.0

queryc0_hammer1.jpg (hammer, subcategory: 81): [2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 1.0

queryc0p_hammer4.jpg (hammer, subcategory: 81): [2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 1.0

queryc0_hammer3.jpg (hammer, subcategory: 81): [2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 1.0

queryc0_pen1.jpeg (pen, subcategory: 210): [2, 2, 2, 1, 2, 2, 2, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 2, 0] - 0.921555063676633

queryc0p_pen1.jpg (pen, subcategory: 210): [2, 0, 0, 0, 0, 0, 0, 0, 2, 0, 1, 2, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 2, 0, 0, 1, 1, 1] - 0.4840746145539654

queryc0p_pen2.jpg (pen, subcategory: 210): [2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 2, 1, 1, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.9385888618696687

queryc0_pen2.jpg (pen, subcategory: 210): [2, 1, 2, 1, 2, 2, 1, 1, 1, 0, 1, 1, 2, 0, 0, 0, 0, 0, 2, 0, 2, 0, 0, 0, 1, 0, 0, 0, 0, 0] - 0.8711222740581338

queryM_nicedayhighlighterblue1.jpg (pen, subcategory: 209): [2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.9340188281353881

queryM_forayhighlighters1.jpg (pen, subcategory: 209): [2, 2, 2, 2, 2, 2, 2, 1, 1, 2, 2, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0] - 0.9910150573409412

queryc0_pen3.jpg (pen, subcategory: 209): [2, 2, 2, 1, 2, 1, 2, 1, 2, 2, 2, 1, 1, 0, 1, 1, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.9585902948965918

queryc0_eyeprotection1.jpeg (eyeprotection, subcategory: 193): [1, 2, 0, 0, 0, 0, 0, 0, 0, 2, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1] - 0.5477921786004996

queryc0_eyeprotection2.jpeg (eyeprotection, subcategory: 193): [2, 2, 2, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0] - 0.9231681859795291

query_safetygoggles1.jpg (eyeprotection, subcategory: 193): [2, 2, 1, 1, 2, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0] - 0.9450592437517717

query_safetyvisor3.jpg (eyeprotection, subcategory: 195): [2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 1, 2, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0] - 0.7909383151725612

queryc0_eyeprotection3.jpg (eyeprotection, subcategory: 195): [2, 0, 2, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 2, 1, 0, 2, 1, 0, 0, 1, 0] - 0.7278291001221053

query_safetyvisor1.jpg (eyeprotection, subcategory: 195): [2, 0, 2, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 2, 1, 0, 0, 1, 1, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.7539256176299036

query_safetyvisor2.png (eyeprotection, subcategory: 195): [2, 0, 0, 0, 0, 2, 0, 0, 0, 0, 1, 1, 1, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 2, 0, 1, 0, 0] - 0.6734406699834796

query_safetyglasses2.jpg (eyeprotection, subcategory: 194): [2, 2, 1, 1, 1, 2, 1, 1, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0] - 0.9101299603427618

queryc0_eyeprotection4.jpg (eyeprotection, subcategory: 194): [2, 2, 1, 0, 2, 1, 2, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0] - 0.9304109555653843

query_safetyglasses1.jpg (eyeprotection, subcategory: 194): [2, 2, 2, 1, 1, 1, 2, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.9523785469864503

Mean NDCG@30: 0.8513906131465765

moondream2 - bge-small-en-v1.5 - Option 3

query_safetyvisor1.jpg (eyeprotection, subcategory: 195): [2, 0, 0, 0, 0, 1, 2, 0, 0, 2, 1, 2, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.7571557036213687

query_safetyvisor2.png (eyeprotection, subcategory: 195): [2, 0, 0, 0, 2, 2, 0, 0, 0, 2, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.788185374895733

query_safetyvisor3.jpg (eyeprotection, subcategory: 195): [2, 0, 1, 0, 2, 0, 0, 0, 0, 1, 1, 1, 0, 2, 1, 0, 1, 2, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.7736091285825393

queryc0_eyeprotection3.jpg (eyeprotection, subcategory: 195): [2, 2, 1, 0, 0, 0, 0, 0, 0, 2, 0, 0, 1, 2, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.8538106675816581

queryc0_eyeprotection2.jpeg (eyeprotection, subcategory: 193): [2, 2, 2, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.9857597569858678

query_safetygoggles1.jpg (eyeprotection, subcategory: 193): [1, 1, 1, 1, 1, 2, 2, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0] - 0.7160105724337945

queryc0_eyeprotection1.jpeg (eyeprotection, subcategory: 193): [2, 0, 0, 1, 0, 0, 0, 0, 0, 0, 2, 0, 1, 2, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0] - 0.6786736855558207

query_safetyglasses2.jpg (eyeprotection, subcategory: 194): [2, 2, 2, 2, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0] - 0.9868591325568985

queryc0_eyeprotection4.jpg (eyeprotection, subcategory: 194): [2, 2, 2, 2, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.9924383389088328

query_safetyglasses1.jpg (eyeprotection, subcategory: 194): [2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.999124769681699

queryc0_anglegrinder3.jpeg (anglegrinder, subcategory: 283): [2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 1.0

queryM_anglegrinder1.jpg (anglegrinder, subcategory: 283): [2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 1.0

queryc0_anglegrinder1.png (anglegrinder, subcategory: 283): [2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 1.0

queryc0_anglegrinder2.jpg (anglegrinder, subcategory: 283): [2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 1.0

queryc0_helmet2.jpg (helmet, subcategory: 198): [2, 2, 2, 2, 0, 2, 0, 2, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.8883907536999223

queryc0_helmet3.jpg (helmet, subcategory: 198): [0, 0, 0, 0, 2, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.18216582889104613

queryM_kaskhelmetwhite.jpg (helmet, subcategory: 198): [2, 2, 2, 2, 0, 2, 2, 2, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.972704987955808

queryc0_helmet1.png (helmet, subcategory: 198): [2, 2, 2, 0, 2, 2, 0, 0, 0, 0, 2, 0, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.9286893150183205

queryc0p_hammer4.jpg (hammer, subcategory: 81): [2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 1.0

queryc0_hammer1.jpg (hammer, subcategory: 81): [2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 1.0

queryc0p_hammer2.jpg (hammer, subcategory: 81): [2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 1.0

queryc0_hammer3.jpg (hammer, subcategory: 81): [2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 1.0

queryc0p_hammer1.jpg (hammer, subcategory: 81): [2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 1.0

queryc0p_hammer3.jpg (hammer, subcategory: 81): [2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 1.0

queryc0_hammer2.jpeg (hammer, subcategory: 81): [2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 1.0

queryM_trousersmodaflame2.jpg (trousers, subcategory: 247): [2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1] - 0.9115101911383411

queryM_trousersmodaflame1.jpg (trousers, subcategory: 247): [0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 2, 0, 2, 2, 2, 0, 2, 0, 0] - 0.3202409790045002

query_flameprooftrousers1.jpg (trousers, subcategory: 247): [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2] - 0.7172422675672576

queryc0_trousers1.jpg (trousers, subcategory: 246): [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1] - 1.0

queryM_trousers1.jpg (trousers, subcategory: 246): [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1] - 1.0

queryc0_trousers3.jpg (trousers, subcategory: 246): [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1] - 1.0

queryM_trousers2.jpg (trousers, subcategory: 246): [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1] - 1.0

queryc0_trousers2.jpeg (trousers, subcategory: 246): [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1] - 1.0

queryM_trousers3.jpg (trousers, subcategory: 246): [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 0, 0, 0] - 0.9682924219799222

query_maskingtape1.jpg (tape, subcategory: 51): [1, 1, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.8499687485047156

query_maskingtape2.jpg (tape, subcategory: 51): [2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.9992863037920513

queryc0_tape1.png (tape, subcategory: 51): [1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 2, 1, 1, 0, 0, 0, 2, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0] - 0.7651495593401194

query_electrictape3.jpeg (tape, subcategory: 52): [2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.9215125277999036

query_electrictape2.jpg (tape, subcategory: 52): [2, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.9504063848835351

query_electrictape1.jpg (tape, subcategory: 52): [2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 1.0

query_garbagetape3.jpg (tape, subcategory: 50): [1, 2, 1, 2, 1, 2, 2, 1, 1, 2, 2, 1, 1, 0, 1, 1, 1, 1, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.8412730992906126

query_garbagetape4.jpg (tape, subcategory: 50): [2, 1, 1, 1, 1, 2, 1, 2, 2, 1, 2, 1, 1, 2, 2, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.8717660368071654

query_garbagetape1.jpg (tape, subcategory: 50): [1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 1, 1, 2, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.7690841886530156

query_garbagetape2.jpg (tape, subcategory: 50): [1, 1, 1, 1, 2, 2, 1, 2, 2, 2, 1, 1, 1, 2, 2, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.7858990465361119

query_packingtape1.jpg (tape, subcategory: 53): [1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 2, 1, 1, 2, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.8006424902617285

queryc0_tape2.jpeg (tape, subcategory: 53): [1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 2, 1, 1, 2, 1, 2, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0] - 0.7421185520809224

query_selfamalgamatingtape1.jpg (tape, subcategory: 53): [1, 1, 2, 1, 1, 1, 1, 2, 1, 1, 1, 1, 2, 1, 1, 2, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.7883124443470987

query_reflectiontape1.jpeg (tape, subcategory: 53): [2, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 2, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0] - 0.9020870238520508

queryc0p_tape2.jpg (tape, subcategory: 54): [2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 1.0

queryc0_tape3.jpg (tape, subcategory: 54): [2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.9258547635327236

queryc0p_tape1.jpg (tape, subcategory: 54): [1, 2, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1] - 0.8720060910173315

queryM_modaflamejacketorange1.jpg (jacket, subcategory: 238): [2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 2, 2, 2, 2, 2, 0] - 0.7465451084440187

queryc0_winterjacket2.jpeg (jacket, subcategory: 238): [0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 2, 2, 1, 2, 0, 1, 0, 0, 1, 2, 2, 2] - 0.18965852414698534

query_flameproofjacket1.jpg (jacket, subcategory: 238): [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1] - 0.10667056813524887

query_genericwinterjacket2.jpg (jacket, subcategory: 239): [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 2, 2, 0, 2, 0] - 0.15334809106229366

queryc0_winterjacket3.jpeg (jacket, subcategory: 239): [2, 2, 2, 2, 2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.6639662715054618

query_genericwinterjacket1.jpg (jacket, subcategory: 239): [2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1] - 0.8664375414601999

query_genericwinterjacket3.jpg (jacket, subcategory: 239): [2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1] - 1.0

query_parkasjacket3.png (jacket, subcategory: 240): [2, 2, 2, 1, 1, 2, 1, 2, 1, 2, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0] - 0.7445680425128463

query_parkasjacket1.png (jacket, subcategory: 240): [2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0] - 0.9742789706052026

queryc0_winterjacket1.jpeg (jacket, subcategory: 240): [2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1] - 1.0

query_parkasjacket2.jpg (jacket, subcategory: 240): [2, 2, 2, 2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.6286255277013445

query_swissknife2.jpg (knife, subcategory: 85): [2, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.9154256239152662

query_swissknife1.jpg (knife, subcategory: 85): [2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.912921559666639

query_pocketknife1.jpeg (knife, subcategory: 85): [1, 1, 2, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.765419338549282

query_pocketknife2.jpg (knife, subcategory: 85): [1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.7536528803212756

queryc0p_knife1.jpg (knife, subcategory: 86): [2, 1, 2, 1, 1, 1, 1, 2, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.8998418688987276

queryc0p_knife2.jpg (knife, subcategory: 86): [2, 2, 1, 1, 2, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.9443838386435541

queryc0_knife2.jpg (knife, subcategory: 86): [2, 1, 2, 1, 1, 1, 1, 1, 2, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0] - 0.9206726695755527

queryc0p_knife3.jpg (knife, subcategory: 86): [2, 2, 1, 1, 1, 1, 1, 2, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.9286671747562876

query_snapoffknife2.jpg (knife, subcategory: 83): [2, 2, 1, 2, 2, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.9772470468290838

query_snapoffknife1.jpg (knife, subcategory: 83): [2, 2, 1, 2, 1, 1, 2, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.9664847287048465

queryc0_knife4.jpg (knife, subcategory: 83): [2, 2, 1, 2, 1, 1, 1, 2, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.9628915316919346

queryc0_knife1.jpeg (knife, subcategory: 84): [0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1] - 0.40025500707433975

queryc0_knife3.jpeg (knife, subcategory: 84): [1, 1, 1, 2, 1, 1, 1, 2, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.7466071365848399

queryM_bahcoknife1.jpg (knife, subcategory: 84): [2, 1, 1, 1, 1, 1, 1, 2, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.8819588969288209

queryc0_paintbrush2.jpg (paintbrush, subcategory: 172): [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 1.0

queryc0p_paintbrush2.jpg (paintbrush, subcategory: 172): [0, 0, 0, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 2, 2, 0, 2, 0, 0, 0, 0, 0, 2, 0, 2, 0, 0, 0] - 0.3146153937177818

queryc0_paintbrush3.jpg (paintbrush, subcategory: 172): [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 1.0

queryc0p_paintbrush1.jpg (paintbrush, subcategory: 172): [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 1.0

queryc0_paintbrush1.jpg (paintbrush, subcategory: 172): [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 1.0

queryc0_winterboots2.jpg (boots, subcategory: 251): [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 0, 2, 2, 0] - 0.9323071422625164

queryc0_winterboots3.jpg (boots, subcategory: 251): [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2] - 1.0

queryc0_winterboots1.jpg (boots, subcategory: 251): [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 0, 0, 2, 0, 0, 0, 0, 2, 2, 0, 0, 0, 2, 2, 2, 2, 2, 2] - 0.7386255561097286

queryc0_pen1.jpeg (pen, subcategory: 210): [2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 0, 2, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0] - 0.9425315208802139

queryc0_pen2.jpg (pen, subcategory: 210): [2, 1, 1, 1, 1, 1, 0, 2, 1, 2, 1, 0, 1, 0, 0, 2, 2, 1, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.8240041110436406

queryc0p_pen1.jpg (pen, subcategory: 210): [2, 1, 1, 1, 1, 1, 1, 2, 2, 1, 0, 0, 1, 0, 2, 0, 0, 2, 0, 0, 0, 2, 0, 1, 2, 0, 0, 0, 0, 0] - 0.8229112076782156

queryc0p_pen2.jpg (pen, subcategory: 210): [2, 2, 2, 2, 2, 1, 1, 2, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 0.8821584653505264

queryM_nicedayhighlighterblue1.jpg (pen, subcategory: 209): [2, 2, 2, 2, 2, 2, 1, 0, 0, 0, 2, 0, 1, 0, 0, 0, 1, 0, 2, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 2] - 0.9105372168536175

queryM_forayhighlighters1.jpg (pen, subcategory: 209): [2, 2, 2, 0, 2, 0, 0, 0, 2, 1, 0, 2, 0, 0, 0, 1, 0, 2, 0, 1, 1, 0, 2, 1, 0, 2, 1, 1, 0, 0] - 0.8808389741529739

queryc0_pen3.jpg (pen, subcategory: 209): [2, 2, 2, 2, 2, 2, 2, 2, 0, 0, 2, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1] - 0.962609354054109

queryc0_notebook1.jpg (notebook, subcategory: 206): [2, 2, 2, 2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 1.0

queryc0_notebook3.jpg (notebook, subcategory: 206): [2, 2, 2, 2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 1.0

queryc0p_notebook2.jpg (notebook, subcategory: 206): [2, 2, 2, 2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 1.0

queryc0p_notebook1.jpg (notebook, subcategory: 206): [2, 2, 2, 2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 1.0

queryc0_notebook2.jpg (notebook, subcategory: 206): [2, 2, 2, 2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] - 1.0

Mean NDCG@30: 0.8590406044434977

Florence-2-large-hf - all-mpnet-base-v2 - Option 1

queryM_anglegrinder1.jpg (anglegrinder, subcategory: 283): True
queryM_kaskhelmetwhite.jpg (helmet, subcategory: 198): True
queryM_trousersmodaflame2.jpg (trousers, subcategory: 247): False
queryM_trousersmodaflame1.jpg (trousers, subcategory: 247): False
queryM_trousers1.jpg (trousers, subcategory: 246): True
queryM_trousers2.jpg (trousers, subcategory: 246): True
queryM_trousers3.jpg (trousers, subcategory: 246): True
queryM_modaflamejacketorange1.jpg (jacket, subcategory: 238): False
queryM_bahcoknife1.jpg (knife, subcategory: 84): True
queryM_nicedayhighlighterblue1.jpg (pen, subcategory: 209): True
queryM_forayhighlighters1.jpg (pen, subcategory: 209): True
Hit Rate: 0.7272727272727273

blip2-opt-2.7b - all-MiniLM-L12-v2 - Option 2

queryM_anglegrinder1.jpg (anglegrinder, subcategory: 283): True
queryM_kaskhelmetwhite.jpg (helmet, subcategory: 198): True
queryM_trousersmodaflame2.jpg (trousers, subcategory: 247): True
queryM_trousersmodaflame1.jpg (trousers, subcategory: 247): True
queryM_trousers1.jpg (trousers, subcategory: 246): True
queryM_trousers2.jpg (trousers, subcategory: 246): True
queryM_trousers3.jpg (trousers, subcategory: 246): True
queryM_modaflamejacketorange1.jpg (jacket, subcategory: 238): True
queryM_bahcoknife1.jpg (knife, subcategory: 84): True
queryM_nicedayhighlighterblue1.jpg (pen, subcategory: 209): True
queryM_forayhighlighters1.jpg (pen, subcategory: 209): True
Hit@10: 1.0

DEPARTMENT OF ELECTRICAL ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY