MediaVerse

A universe of media assets
and co-creation opportunities

D3.2

Content Discovery and Recommendation, Annotation and Adaptation Framework

Project Title	MediaVerse
Contract No.	957252
Instrument	Innovation Action
Thematic Priority	ICT-44-2020 Next Generation Media
Start of Project	1 October 2020
Duration	36 months

Deliverable title	Content discovery and recommendation, annotation and adaptation framework
Deliverable number	D3.2
Deliverable version	V1.0
Previous version(s)	D3.1
Contractual Date of delivery	30.06.2022
Actual Date of delivery	30.06.2022
Nature of deliverable	Other
Dissemination level	Public
Partner Responsible	LINKS Foundation
Author(s)	Federico D'Asaro, Sara De Luca, Lorenzo Bongiovanni, Giuseppe Rizzo (LINKS), Christos Koutlis, Ioannis Sarridis, Panagiotis Galopoulos, Manos Schinas, Manolis Krasanakis, Dimitris Karageorgiou, Nikolaos Sarris, Symeon Papadopoulos (CERTH), Antonio Calvo (ATOS), Stephan Gensch (VRAG)
Reviewer(s)	Manos Schinas (CERTH), Rahel Luder, Robin Ribback (STXT), Stephan Gensch, Tino Breddin (VRAG)
EC Project Officer	Alberto Rabbachin

Abstract	This deliverable comprises software modules and accompanying reports for three MV parts: 1) a component for new media content annotation and understanding, 2) a component for retrieving and recommending multimodal content, 3) a component for adapting the content to the MediaVerse ecosystem.

Keywords	Artificial Intelligence, Media Annotation, Media Retrieval, Recommender System, Transcoding

Copyright

MediaVerse is an H2020 Innovation Project co-financed by the EC under Grant Agreement ID: 957252.

Revision History

VERSION	DATE	MODIFIED BY	COMMENTS
V0.1	22/04/2022	Giuseppe Rizzo (LINKS)	First draft of the table of contents
V0.2	15/05/2022	Federico D'Asaro (LINKS)	Edit Cross-modal Retrieval Technology (Section 3)
V0.3	23/05/2022	Sara De Luca (LINKS)	Edit Multimodal Recommendation Technology (Section 4)
V0.4	25/05/2022	Christos Koutlis, Ioannis Sarridis, Panagiotis Galopoulos, Manos Schinas, Manolis Krasanakis, Dimitris Karageorgiou, Nikolaos Sarris, Symeon Papadopoulos (CERTH)	Edit New media Understanding and Annotation Service (Section 2)
V0.5	26/05/2022	Antonio Calvo (ATOS)	Edit Services for the automatic adaptation of the MediaVerse framework content (Section 5)
V0.6	30/05/2022	Lorenzo Bongiovanni (LINKS)	Draft introduction and internal revision
V0.7	03/06/2022	Manos Schinas (CERTH)	1st review
V0.8	07/06/2022	Giuseppe Rizzo (LINKS)	Provision of the executive summary, introduction, conclusion
V0.9	07/06/2022	Stephan Gensch (VRAG)	Add OMAF transcoding (Section 5)
V1.0	30/06/2022	Giuseppe Rizzo, Federico D'Asaro, and Sara De Luca (LINKS), Robin Ribback (STXT), Stephan Gensch (VRAG), Evangelia Kartsounidou, Symeon Papadopoulos, Manos Schinas (CERTH)	Final review and quality control

Glossary

ABBREVIATION	MEANING
GDPR	General Data Protection Regulation
HMD	Head-mounted Display
WP	Work Package
MV	MediaVerse
DAM	Digital Asset Management
ANN	Artificial Neural Network
CNN	Convolutional Neural Network
RPC	Remote Procedure Call
IR	Information Retrieval
CMR	Cross Modal Retrieval
LSTM	Long Short-Term Memory
CLIP	Contrastive Language-Image Pre-Training
FAISS	Facebook AI Similarity Search
MSCOCO	Microsoft Common Objects in Context
ViT	Vision Transformer
MRR	Mean Reciprocal Rank
RS	Recommender System
ILD	Intra-List Distance
HDBSCAN	Hierarchical Density-Based Spatial Clustering of Applications with Noise
TSNE	t-distributed stochastic neighbour embedding
IPFS	InterPlanetary File System
OMAF	Omnidirectional Media Format

Table of Contents

Index of Figures

Index of Tables

Executive Summary

This deliverable reports on the work carried out in the MediaVerse WP3 - Next Generation Content Management, Understanding and Interlinking, documenting the Content Discovery and Recommendation, Annotation and Adaptation Framework. It presents the work conducted until M21 by presenting the design, development, testing and deployment of a framework that provides a) content annotation, discovery, and retrieval - by using the annotations generated – as well as b) recommendations and c) adaptation of contents. These technologies have been developed to be modular microservices and integrated into the MediaVerse ecosystem.

The annotation system answers the research question of how to add metadata to content shared on the MediaVerse platform in order to ease retrieval and management. The retrieval system addresses the research question of how to look for multimodal content relying on both the generated annotations and commonalities among visual and textual clues detected from the content. The recommender system addresses the research question of how to support users of the platform to consume media by receiving suggestions based on similar content than those already seen. Finally, the adaptation framework addresses the research question of how to adapt content for distribution across and beyond the MediaVerse platform.

LINKS led the conceptualisation of this deliverable, the editing of Section 3 and 4, Summary, Introduction and Conclusion, CERTH led the development and reporting of the annotation service in Section 2 and ATOS led the development and reporting of the media adaptation framework of Section 5.

1 Introduction

Search and *Recommendation* are two important functionalities for digital asset management systems. The first allows a user to formulate a *query* and retrieve the corresponding most relevant results out of all content present inside the media platform, while the second aims to automatically suggest personalized content to a user based on his/her own interests.

When it comes to *Search*, even though the query is usually formulated by the user in the form of text, the retrieved content should belong, in principle, to any type of media supported, if it is most relevant to the query, i.e., results should include images, videos. This requirement puts emphasis on the Search models to be able to operate in a *shared semantic space*, where content from all supported media types can be compared to each other and with the user query. To achieve this shared space of all media types there are at least two avenues.

The first, discussed in Section 2, is to express all media into the same space where the query is formulated (i.e., the text space), reducing the problem to a text search. This approach has the advantage of being able to exploit all the tools that are available for text search, most of which are very fast and have good performance, while it has the downside of having to factor in the loss in information and accuracy that comes by reducing a different media type into text. Section 2 describes the media annotation tool that can produce accurate text annotations for any kind of media. The annotations not only describe objects present in the media but also the actions happening, and the faces depicted if those correspond to celebrities. Once these annotations are produced, they are used to represent the content in any text search.

A second way, discussed in Section 3, is to embed content from different kinds of media directly into the same *cross-modal semantic space* where content becomes directly comparable in terms of Euclidian distance. This approach has the advantage of being able to capture the semantics of the content more comprehensively; however, it is harder to integrate with other existing metadata since there are no available search tools built explicitly around that. In Section 3, we describe the development of a cross-modal retrieval system leveraging a state-of-the-art cross-modal embedder and a very fast and scalable vector similarity search tool.

Recommendation is another very important functionality to have in a social media. The goal is to improve the overall experience by suggesting relevant and novel content to the user autonomously. Most modern recommendation systems leverage the historical behaviour of the users of the social media, to recommend a content that a user has not seen but another user with close interests has. In MediaVerse, however, we do not want to centrally monitor user behaviours, therefore, in Section 4, we describe our effort to build a Recommender system that relies on contents posted by the same user, bypassing the need to centralize user behaviours on the platform. Similar to the retrieval system, the employed technology is based on cross-modality, so that different kinds of media can be suggested to the user. We present a method to generate seeds from a user's previous posts and evaluate the performance in terms of serendipity of the recommended contents.

Finally, Section 5 describes the Content Adaptation framework for the ingestion, manipulation, and sharing of content for all supported media types. One goal is to optimize the data format to for the most modern network technologies (i.e., 5G). An additional goal is to provide transformations that support low bandwidths and different kinds of screens: smartphones, tablets for the platform to be usable by any kind of user. After providing a detailed description on state-of-the-art content adaptation methods in previous deliverables, here we focus on the approach and tests made on real content and the transformation methods and integration for new media formats (i.e., 3D models).

2 New Media Understanding and Annotation Service

This section describes the work conducted in T3.1. It mainly relates to automatic understanding and annotation of digital assets stored in MediaVerse nodes, such as images, videos and 3D content. Additionally, we provide details regarding the building of the API that supports the annotation service and the user-friendly model building service that will provide amateur users with tools, which will enable them to create new models that fit their own needs. Figure 1 depicts the interactions between the MediaVerse node's modules and the annotation service. When a user uploads an asset (image, video or 3D model) through the gateway and the UI, the Digital Asset Management module (DAM) call the Annotation service to obtain the annotations described in the following sections. These annotations are then indexed in the node's index (Apache SOLR) to facilitate search and retrieval. At retrieval time, the user can issue filter parameters alongside the text query to find the most relevant assets. For example, for a user that needs an image depicting at least one person and a dog, but not being a meme image, the search module can generate the appropriate SOLR query that leverages image annotations such as meme detection (Section 2.1.1) and object detection. In a similar manner, the search module can leverage video and 3D content annotations.

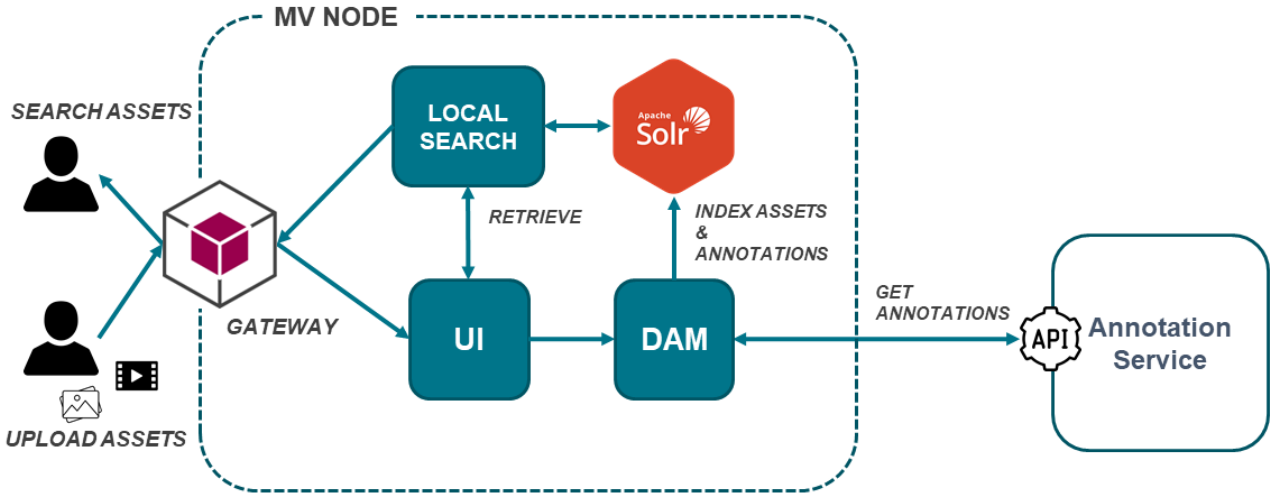


Figure 1: Interactions between the MediaVerse node's modules and the annotation service.

2.1 Images

In 2.1, we elaborate on updated as well as new models deployed on our service that support image annotation.

2.1.1 Meme Detection

In this sub-section, we describe a methodology that improves the previously deployed model for Internet image meme detection proposed in D3.1 - Next Generation Content Model and Algorithms for New Media Types[1]. The new method utilises the visual part of image memes as instances of the regular image class and the initial image memes as instances of the image meme class to force the model to concentrate on the critical parts that characterise an image meme. Additionally, we employ a trainable attention mechanism on top of a standard ViT

[1] https://mediaverse-project.eu/wp-content/uploads/2021/10/MediaVerse_D3.1_NextGeneration-ContentModel-and-Algorithms-for-NewMediaTypes.pdf

architecture (Dosovitskiy et al. 2020) to enhance the model's ability to focus on these critical parts and make the predictions interpretable. We call the corresponding model MemeTector. The findings indicate that light visual part utilisation combined with sufficient text presence during training provides the best and most robust model, surpassing state of the art. This work was submitted for publication to a relevant journal.

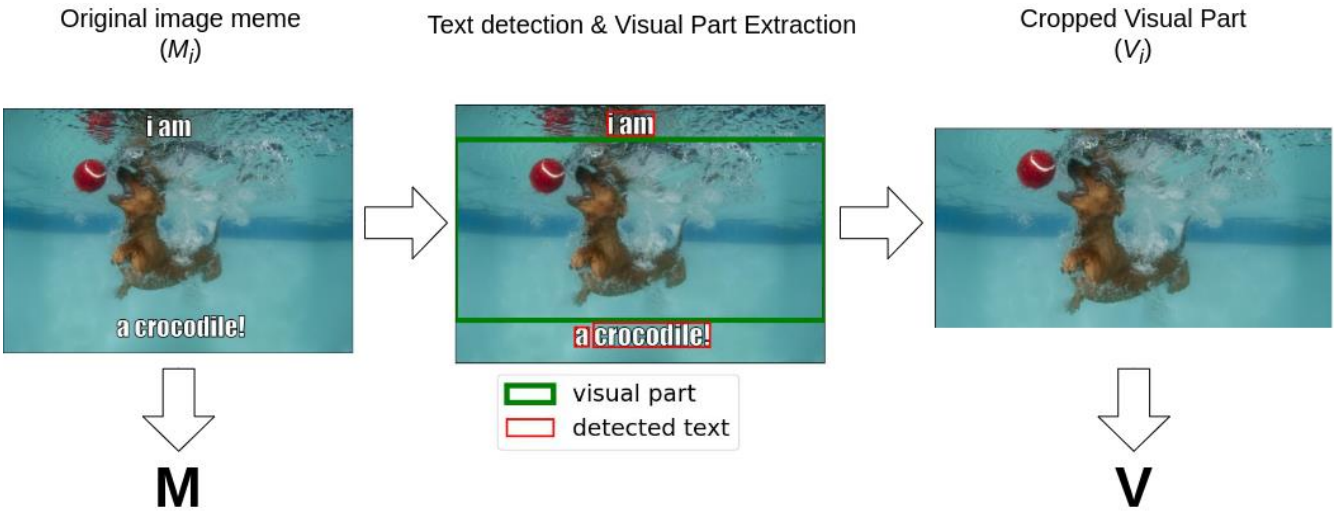


Figure 2: The Visual Part Utilisation process.

Figure 2 illustrates the visual part utilisation process, which creates two sets of instances: one with image memes, and one with the corresponding visual parts[2]. Figure 3 presents Algorithm 1, which describes the extraction part of the algorithm. TextFuseNet (Ye, 2020) is a text detection deep neural network outputting a set of bounding boxes that correspond to text locations in the input image. The letter *p* is the percentage of the initial image's area, which equals the rectangle's area; *r* is the rectangle's aspect ratio; f_W and f_H are the percentages of the initial image's width *W* and height *H,* which define the rectangle's centre.

Algorithm 1: Visual part extraction

input : M_i
output: V_i
$W, H \leftarrow size(M_i)$;
$\mathbf{B}_i \leftarrow \text{TextFuseNet}(M_i)$;
$\mathcal{R} \leftarrow \emptyset$;
for $p \in [0.1, 0.9]$, $r \in \left[\sqrt{p}, \frac{1}{\sqrt{p}}\right]$, $f_W \in \left[\frac{\sqrt{p}}{2r}, 1 - \frac{\sqrt{p}}{2r}\right]$, $f_H \in \left[\frac{r\sqrt{p}}{2}, 1 - \frac{r\sqrt{p}}{2}\right]$ **do**
  $R \leftarrow \left(f_W \cdot W - \frac{W\sqrt{p}}{2r}, f_H \cdot H - \frac{\sqrt{p} \cdot H \cdot r}{2}, f_W \cdot W + \frac{W\sqrt{p}}{2r}, f_H \cdot H + \frac{\sqrt{p} \cdot H \cdot r}{2}\right)$;
  if $R \cap B = \emptyset, \forall B \in \boldsymbol{B}_i$ **then**
    $\mathcal{R} \leftarrow \mathcal{R} \cup \{(R, p)\}$;
$p_{max} \leftarrow \max\left(\{p \mid (R, p) \in \mathcal{R}\}\right)$;
$R_{V_i} \leftarrow RandomSample\left(\{R \mid ((R, p) \in \mathcal{R}) \wedge (p = p_{max})\}\right)$;
$V_i \leftarrow crop(M_i, R_{V_i})$;

Figure 3: Meme visual part extraction algorithm.

We utilise the extracted visual parts V_i of image memes M_i as regular image instances in order to force the model's focus on the critical parts that discriminate them. More precisely, we consider the set $M = \{M_i\}_{i=1}^{k}$ that contains image memes and the set $V = \{V_i\}_{i=1}^{k}$ that contains the corresponding visual parts. To assess the extent

[2] The original image M_i, which belongs to the set of image memes M, is passed through the visual part extraction algorithm that identifies the corresponding visual part Vi, crops it and adds it to the set V.

to which VPU is useful, we also conduct experiments mixing instances of V and web-scraped regular images for the construction of regular images class R. Additionally, given the inherent text presence in image memes another crucial aspect to consider is the extent to which text presence in regular images affects the model's robustness. Hence, we consider two more sets as pools for R construction, namely $R_p = \{R_i^p\}_{i=1}^k$ for web-scraped regular images with text presence and $R_a = \{R_i^a\}_{i=1}^k$ for web-scraped regular images with text absence. The model's objective is to correctly classify the instances of the two sets, M and R, with:

$$R = \{V_i\}_{i=1}^{k\cdot(1-P_W)} \cup \{R_i^p\}_{i=1}^{k\cdot P_W\cdot P_T} \cup \{R_i^a\}_{i=1}^{k\cdot P_W\cdot(1-P_T)} \quad (1)$$

where P_W and P_T denote the fraction of web-scraped regular images out of the total number of regular images and the fraction of web-scraped regular images with text presence out of the total number of web-scraped regular images, respectively. It is important to note that with this formulation M and R preserve the same cardinality k. For more clarity, we list the utilized and generated sets along with a description in Table 1.

Table 1: Description of utilized sets of images for the task of meme detection.

SYMBOL	DESCRIPTION
M	set of image memes that defines the class
V	set of extracted visual parts from set M
R_a	set of regular web-scraped images without text
R_p	set of regular web-scraped images with text
R	set of regular images that defines the class, it is a combination of parts from V , ^{R}a and ^{R}p

The proposed model architecture expands upon a standard ViT by adding an attention module that compares the last layer's class token's embedding y with the patch embeddings of previous layers $\{z_l^{1:N}\}$ with l being the layer index and N the number of patches. More precisely, we compute the compatibility score:

$$s_l^i = v \times [y; z_l^i] \quad (2)$$

where $i \in \{1,\ldots,N\}$, $l \in \{1,3,\ldots,n\}$, $[\cdot;\cdot]$ denotes concatenation, $\times$ denotes dot product, and v is a trainable vector. Then, attention weights are calculated by applying the softmax function and the context vectors per layer are the weighted average of the layer's patch embeddings. The concatenation of all context vectors is processed by three dense layers for the final prediction, the first two are GELU activated and the last has one sigmoid unit.

For the image meme class and the visual parts set we consider the Facebook's Hateful Memes dataset[3], while as a pool for the regular images sets, we consider the Google's Conceptual Captions dataset[4]. As a starting point for the image meme class M, we consider the 10,000 instances of the Hateful Memes dataset. Then, we extract the visual parts of these image memes resulting in 9,984 images considered as regular to form the set V. The mean area fraction across all $V_i \in V$ is 64.3\%. For the remaining 16 images, the VPU algorithm was unable to find a rectangle with no overlap with text. Thus, for all four sets M, V, R_p and R_a we consider the same size of k=9,984 instances. To do so, we discard the same 16 image memes from M and sample from Google's Conceptual Captions dataset k=9,984 instances to form R_p and another k=9,984 instances to form R_a, respectively. For sample mixing, to form the class of regular images R we only need to determine P_W and P_T. Also, to assess the impact of both VPU and text presence in the model's performance, we consider several synthesis scenarios $S_i = (P_W, P_T)$, with $i = 1,\ldots 13$, for R, presented in Table 2.

[3] https://ai.facebook.com/blog/hateful-memes-challenge-and-data-set/
[4] https://ai.google.com/research/ConceptualCaptions/

Table 2: Synthesis scenarios for regular images class construction based on P_W and P_T.

P_W	0%	33%				67%				100%			
P_T	0%	0%	33%	67%	100%	0%	33%	67%	100%	0%	33%	67%	100%

Furthermore, we consider the same scenarios both on the training and test sets and experiment with crossed scenarios (S_i, S_j), e.g., the model is trained on $S_1 = (P_w = 0\%, P_T = 0\%)$ but evaluated on $S_{13} = (P_w = 100\%, P_T = 100\%)$, resulting in 13·13=169 crossed scenarios to analyse. For sample splitting, we initially select 85% training, 5% validation and 10% test samples for each set M, V, R_p and R_a, and construct R per split. For M and V, we consider the same index split in order not to include the visual part V_i in one split (e.g., training), and the initial image meme M_i in another (e.g., test). The training and validation sets always derive from the same scenario S_i, while the evaluation is performed on all test scenarios S_j.

We also compare our approach to other state of the art image classification models fine-tuned for the task. These competitive models are ViT, EfficientNetB5, ResNet50, and VGG16. After an ablation analysis concerning the best training configuration, we opt for the model trained on $S_9 = (P_w = 67\%, P_T = 100\%)$ which from now on will be referred to as MemeTector. For comparison with the state-of-the-art models, we provide Tables 3 and 4 in which the superior performance of MemeTector is illustrated.

Table 3: Models' performance in terms of average binary accuracy across all crossed scenarios.

MODEL	ACCURACY
VGG16	91.36%
ResNet50	92.31%
EfficientNetB5	90.05%
ViT	94.17%
MemeTector	**94.98%**

Table 4: Fraction of crossed scenarios that the MemeTector model surpasses competitive models.

MODEL	FRACTION
VGG16	147/169=86.98%
ResNet50	109/169=64.50%
EfficientNetB5	162/169=95.86%
ViT	148/169=87.57%

2.1.2 Action Recognition

The task of action recognition is typically considered for videos. Section 2.2.1 elaborates our work on video action recognition related to the annotation service. However, images may also contain static actions that could be recognized and that would provide the user with valuable context. For instance, typical image-oriented models would just recognize the person, the racket, and the ball in the photo of Roger Federer depicted in Figure 4. By applying action recognition, the corresponding action ("playing tennis") could be detected and used in asset retrieval. To this end, we consider the ResNet152 deep neural network trained on the Kinetics400 dataset[5].

[5] Some examples are cleaning pool, crying, driving car, eating chips, parkour, playing guitar, reading book and rock climbing.

Figure 4: Roger Federer playing tennis.

2.1.3 Disturbing Content Detection

Disturbing content on images refers to content that depicts humans or animals subjected to violence, harm, and suffering that can cause feelings of worry, concern, or anxiety to the viewer. In many professions, such as journalism, employees are often exposed to content generated by users, and inevitably, they are exposed to disturbing content that could cause emotional traumas. For instance, Russia's invasion of Ukraine resulted in the generation of a large amount of such content that journalists face daily. Therefore, developing automated methods to detect disturbing visual content is of utmost importance. Such a model has been built in the framework of T5.1 regarding content moderation and is now part of the media annotation service. More details on the model building and experiments will be available in D5.4 - Content moderation toolset.

2.2 Videos

In 2.2, we elaborate on the models that are deployed on our service and support video annotation. In addition to the typical video-oriented models, we present repurposed models primarily built for image annotation that we utilise to extract further information from video frames.

2.2.1 Action Recognition

Related Work

Video classification is an active field of study that has attracted great attention lately. The initial attempts consider 3D convolutions as an extension to 2D convolutions, present in the well-known image classification architectures to take into account the time dimension of videos in the processing, and take advantage of large receptive fields for long-term dependencies (Ji et al. 2013; Karpathy et al. 2014; Tran et al. 2015). A tweak proposed by Carreira and Zisserman (2017) to inflate the ImageNet pre-trained weights as a starting point for training 3D convolutions resulted in state-of-the-art performance of the widely used I3D architecture. In addition, other simpler models performing aggregation of frame-level predictions have been proposed (Wang et al. 2016). Two-stream networks have also been employed for action recognition either using single frames as input to the first stream and the optical flow as input to the second stream (Simonyan & Zisserman 2014) or using a slow stream of low frame rate and a fast stream of high frame rate, but with lower channel capacity, namely the widely adopted SlowFast architecture (Feichtenhofer et al. 2019). Moreover, the recently proposed X3D architecture (Feichtenhofer 2020) achieves great performance by expanding a tiny 2D image classification architecture along multiple network axes, in space, time, width and depth. However, although all architectures until 2020 are based on convolutions, in 2021 the state of the art is overwhelmed by transformer-based

architectures that expand the Vision Transformer (ViT) (Dosovitskiy et al. 2020) for images. For instance, ViViT investigates four different architectures to handle the spatial and temporal information either by processing all patches at once or by separating the spatial from the temporal analysis of the signal in multiple ways (Arnab et al. 2021). Also, Bertasius et al. (2021) study five space-time self-attention schemes to finally propose TimeSformer architecture. Finally, Video Swin transformers utilise the Swin Transformer architecture (Liu et al. 2021a) and expand it to videos achieving the current top-1 accuracy on multiple benchmarks (Liu et al. 2021b).

Pre-trained Models

To produce tags for videos we rely on four state-of-the-art pre-trained deep learning architectures for video classification: I3D, SlowFast, TimeSformer and Video Swin Transformer. We consider three python libraries, namely GluonCV[6], MMAction2[7] and Video-Swin-Transformer[8], which make these models publicly available to compare them with regard to outcome quality as well as computational cost at inference time. More precisely, we consider I3D inception V1, SlowFast 4x16 ResNet50, TimeSformer divST 8x32x1 and Video Swin Transformer tiny version all pre-trained on Kinetics-400 (Kay et al. 2017) dataset that categorises videos in 400 different activities[9].

Video Processing Pipeline

The videos uploaded in MV nodes are of varying length, which can exceed seconds. Most video classification datasets contain videos of a few seconds and while theoretically the corresponding pre-trained models can handle any length, in practice large videos are intractable for most GPUs. Additionally, a video might contain several parts of different nature and each part could potentially be classified differently from the others. Having these two things in mind, we consider a very simple heuristic pipeline consisting of video splitting before the processing part and tag aggregation after it, to avoid model confusion as well as computational inefficiency.

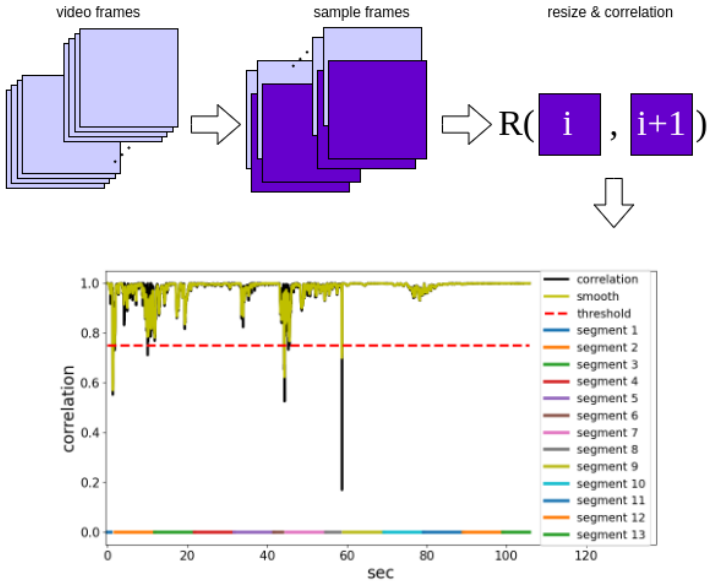


Figure 5: Video splitting.

[6] https://cv.gluon.ai/

[7] https://github.com/open-mmlab/mmaction2

[8] https://github.com/SwinTransformer/Video-Swin-Transformer

[9] https://deepmind.com/research/open-source/kinetics

In Figure 5, we present the first stage of the pipeline namely video splitting in coherent parts. First, we uniformly downsample and resize the frames (these are not mandatory steps, but they reduce the computation time without significant accuracy loss - here we resize to 50x50 but do not downsample) and then we calculate Pearson correlation between consecutive frames (flattened). Additionally, we smooth the correlation trajectory with the moving average filter of order 2, to avoid random spikes. To identify big discrepancies that potentially are transition points, we set a threshold and extract the corresponding video segments between the transition points. Then, if the segments are larger than 10 seconds, we re-segment them to have a maximum length of 10 seconds. Finally, for the second stage of our pipeline, namely tag aggregation, after passing each segment to the video classification model we end up with multiple tags from which we finally keep only the unique ones.

In that way, (a) we do not process video segments that contain diverse semantics, which might confuse the model, (b) we keep the computational cost at affordable levels, and (c) we obtain multiple tags per video if it contains multiple parts of different nature.

Experimental Results

We consider three example videos as input to our pipeline to assess its efficiency with respect to using different video classification models. The first is a news video template[10] containing multiple distinct temporal segments of different activities (Figure 6), the second is a video recording of an aeroplane being on fire after a crash (Figure 7), and the third is a short video showing tea pouring (Figure 8). Table 5 provides information for each video, such as the duration and the number of segments they are split for processing. After passing each input video to our pipeline, we obtain the aggregated outputs of each model as well as duration of video loading[11], model loading and annotation, and illustrate the corresponding video classification model comparison, in Table 6[12].

Figure 6: Indicative sample frames of the three example input videos: (a) news video template

Figure 7: Indicative sample frames of the three example input videos: (b) aeroplane crash

[10] https://www.youtube.com/watch?v=06vH-Uakx-w

[11] This is not a metric used for model selection but its value affects the total duration so we illustrate it for clarity.

[12] All experiments are conducted on a NVIDIA GeForce GTX 1080 GPU.

Figure 8: Indicative sample frames of the three example input videos: (c) tea pouring

Table 5: Example video statistics.

	DURATION	SEGMENTS
news video template	24.0 sec	6
aeroplane crash	105.9 sec	11
tea pouring	7.7 sec	1

Table 6: Video classification models comparison in terms of produced tags and time consumption.

		MODELS			
		I3D	SlowFast	TimeSformer	Swin
Kinetics-400 top-1 accuracy*		71.8%	75.3%	77.92%	**78.8%**
News video template					
tags produced for the input demo video		extinguishing fire 95% sailing 64% marching 34% answering questions 15% stretching arm 2%	extinguishing fire 100% news anchoring 75% marching 53% sailing 39% dancing ballet 4% stretching leg 2%	extinguishing fire 99% marching 54% answering questions 38% sailing 26% exercising arm 7%	extinguishing fire 99% sailing 72% playing didgeridoo 34% answering questions 22% juggling balls 1%
duration	load video	1.07 sec	1.08 sec	1.03 sec	1.00 sec
	load model	**0.29 sec**	0.48 sec	2.21 sec	0.38 sec
	annotation	7.63 sec	3.87 sec	**1.98 sec**	23.02 sec
	total	8.99 sec	5.43 sec	**5.22 sec**	24.4 sec
Aeroplane crash					
tags produced for the input demo video		extinguishing fire 60% flying kite 28%	hurdling 97% extinguishing fire 68% abseiling 49%	extinguishing fire 97%	using remote controller (not gaming) 71%

		marching 15% driving car 14%	water skiing 21% driving car 17% flying kite 8%	using remote controller (not gaming) 74% flying kite 42% skydiving 29%	extinguishing fire 63% driving car 49% archery 30% flying kite 21% playing paintball 18%
duration	load video	1.95 sec	1.93 sec	1.95 sec	1.84 sec
	load model	**0.29 sec**	0.58 sec	2.45 sec	0.40 sec
	annotation	43.43 sec	9.80 sec	**4.02 sec**	47.11 sec
	total	45.67 sec	12.31 sec	**8.42 sec**	49.35 sec
Tea pouring					
tags produced for the input demo video		making_tea 72%	making_tea 93%	making tea 99%	folding napkins 25%
duration	load video	2.86 sec	3.11 sec	3.07 sec	2.79 sec
	load model	**0.31 sec**	0.59 sec	2.35 sec	0.47 sec
	annotation	2.92 sec	1.01 sec	**0.41 sec**	9.66 sec
	total	6.10 sec	**4.72 sec**	5.84 sec	12.92 sec

*Note: *Refers to the models that we used, not to the best results presented in the papers.*

It seems that all models produce meaningful tags for the videos of the example; however, some produce wrong labels (in most cases with low confidence). Although the video swin transformer is the best performing model on Kinetics-400, it fails to classify the tea pouring video to something similar at least as the other models did. In addition to that, it requires video splitting with third party libraries, which makes the whole process very slow. On the other hand, TimeSformer architecture not only provides quality tags for the example videos, but also it is the fastest model in video processing. A downside of TimeSformer is that it is slow in model loading, but in an efficient interface, the model loading is done once before processing multiple videos. Comparing SlowFast with TimeSformer we observe that it provides equivalent annotation quality, being 4-5 times faster at loading but 2 times slower at processing. Finally, the I3D architecture, which is still relevant but a bit outdated, provides relevant tags and its loading is very fast, but video duration severely affects the processing time.

Deployment Issues

There is nowadays a wide variety of deep learning frameworks, each offering a unique view about how training and inference should be. However, for efficient model deployment, inference should not be tied to the framework that the model was initially created with, and even more generally should not be tied to the Python runtime. For this purpose, each framework has devised each own independent description format for their models; for example, Tensorflow has the SavedModel format and PyTorch has torchscript programs. ONNX fits the same landscape but did not originate from any framework; rather it is described as an open format built to represent machine learning models. Inference with any of these tools provides interoperability, performance benefits through optimizations, ability to invoke the models from languages like C instead of Python, and integration with existing tools like NVidia's Triton Inference Server, which the Media Annotation Server internally

uses for model inference. In summary, all of these are compelling reasons to require the export of the models of the Media Annotation Server in one of the aforementioned formats.

Following the conclusions of our experiments, even though we initially selected the TimeSformer model for deployment, we realised that exporting it to ONNX format (or any other framework-agnostic format) is not yet supported by MMAction2. Consequently, we headed for our second-best option, namely the SlowFast architecture, for which similarly, GluonCV does not support ONNX exporting at the moment (while there are GitHub issues for this matter so it might be considered for the future). Fortunately, we encountered its implementation with pre-trained weights on Kinetics-400 provided by PyTorch[13] that we then exported to ONNX format without any issues. Hence, we opt for it to be deployed in the Media Annotation API for video action recognition purposes. However, it is important to mention a shortcoming of this implementation: according to our experimentation up until now, it is not feasible to provide inputs with a different number of frames than the default one, which we alleviated by uniform sampling. Table 7 presents the corresponding annotations and performance metrics of the deployed model on the three example videos. Additionally, the declared top-1 accuracy that it achieves on the Kinetics-400 dataset is 76.94%.

Table 7: ONNX exported SlowFast PyTorch implementation performance metrics.

DEMO VIDEO NAME		NEWS VIDEO TEMPLATE	AEROPLANE CRASH	TEA POURING
tags produced for the input demo video		extinguishing fire 100% marching 72% crossing river 59% answering questions 15% dancing ballet 7%	extinguishing fire 98% flying kite 96% diving cliff 89% passing American football (not in game) 39% driving car 31% motorcycling 20% bungee jumping 17%	making tea 98%
duration	**load video**	1.10 sec	2.26 sec	2.68 sec
	load model	0.14 sec	0.13 sec	0.13 sec
	annotation	5.83 sec	8.10 sec	0.76 sec
	total	7.07 sec	10.49 sec	3.57 sec

2.2.2 Expanding Image Models to Support Video Input

Videos may contain objects, well-known persons, and disturbing scenes as well as images do. Hence, applying the corresponding image models of our service on video assets will provide further useful annotations. Of course, applying the models on all frames of input videos would be redundant as well as catastrophic in terms of service latency. Therefore, we consider temporal segmentation methods to select key-frames for analysis to annotate the video.

[13] https://pytorch.org/hub/facebookresearch_pytorchvideo_slowfast/

Object Detection

Coherent-frame video parts are likely to contain the same objects. Based on that assumption, we first segment the input video into scenes using the method introduced in *Video Processing Pipeline section* and then randomly select one frame per scene. The object detection model's output labels for several frames are finally returned along with the corresponding timestamps and confidence.

Celebrity Recognition

Following a similar path for face recognition, one frame per second is first selected using uniform random sampling. Then, face extraction based on MTCNN is performed per selected frame. Finally, after applying the VGGFace2 celebrity recognition model on all the cropped parts containing the extracted faces, it returns the names of all identified persons along with the corresponding timestamps and confidence. Note that if a person is identified multiple times in the video all occurrences are returned with different timestamps.

Disturbing Content Detection

Similarly to the previous tasks, in disturbing content detection we exploit the method presented in *Video processing pipeline section* to segment the input video into several scenes. Then, we keep one frame per scene, considering that the scene's frames share similar context. Afterwards, we classify these frames as disturbing or non-disturbing by the model described above. Given that each frame represents a scene, it returns information about the scenes classified as disturbing. This information comprises the timestamp the scene starts, the end-timestamp, and the corresponding confidence.

2.3 3D Content

As described in the deliverable D3.1 - Next Generation Content Model and Algorithms for New Media Types[14], the annotation process of 3D scenes is inextricably connected to the representation of the 3D data. MediaVerse aims to cover a wide variety of representations. This requirement has driven our decision with respect to the employed annotation model. In our initial efforts, we exploited BAAF-Net (Qiu et al., 2021), which is a State-of-the-Art (SotA) network addressing the problem of semantic segmentation for indoor scenes. However, as any modern Artificial Neural Network (ANN) that deals with 3D data, there are several limitations that accompany BAAF-Net, with the most important being the lack of generalisation capabilities (Guo et al., 2020). This is mainly attributed to the lack of available public datasets for the corresponding task. For instance, S3DIS (Armeni et al., 2016) and ScanNet (Dai et al., 2017) contain 13 and 21 classes, respectively. It becomes evident that the aforementioned raise an obstacle for the goals of MediaVerse, since end-users are context-unaware 3D content creators. Another important limitation concerns the inference time of SotA ANN models for 3D data, making them inadequate for real time 3D scene annotation. Finally, the last identified limitation is related to the fact that a single ANN cannot perform single 3D object annotation and 3D scene annotation. To achieve reliable annotation results for both 3D objects and 3D scenes, we should employ a different ANN for each task.

Having all the above in mind, we followed a different course of action during the second phase of the project. More specifically, we focused on methodologies that can provide annotations in a broader range, reaching beyond the dataset that the network was trained on.

[14] https://mediaverse-project.eu/wp-content/uploads/2021/10/MediaVerse_D3.1_NextGeneration-ContentModel-and-Algorithms-for-NewMediaTypes.pdf

2.3.1 Multi-View Object Detection

In the early stages of 3D annotation research, the focus was mainly on transforming the 3D data into 2D[15]. Such a transformation allowed researchers to leverage the tools created for the 2D image processing domain (which were of higher maturity). Among the existing practices in this direction, rendering the 3D scene to multi-view images appears as the most promising solution since it allows employing 2D Convolutional Neural Networks (CNN) on each of the rendered images. The work of (Su et al., 2015) appears to be the first milestone in multi-view 3D processing. The introduced idea is simple yet powerful, as it can exploit the inherent characteristics of CNNs. Initially, a 3D shape is rendered by several different virtual cameras (e.g., simulating different perspectives of the same shape). Then, the obtained 2D images are analysed by a particular CNN to extract view-specific features. These are subsequently pooled across views and passed through a CNN to obtain a compact shape descriptor. Such an approach enables the employment of the same network for both 3D object classification and retrieval. Since the inaugural work of Su et al. (2015), several works (Kundu et al., 2020; Jaritz et al., 2019; Dai et al., 2018) have built on top of it to provide solutions for a wide variety of computer vision tasks (e.g., object classification, semantic segmentation, scene understanding, etc.).

Following the paradigm of Su et al. (2015) and with 3D annotation as the ultimate task, we employ the multi-view approach for three reasons. Firstly, we can take advantage of the ANNs that have been already deployed for image annotation (object detection, caption etc.). Moreover, we can benefit from all the developments of the 2D image processing domain, which is by far more studied than the 3D processing one. Finally, it gives MediaVerse the opportunity to cover a wider spectrum of concepts, given that public datasets for image annotation offer thousands of concepts (e.g., ImageNet).

2.3.2 Framework for Multi-View Object Detection

The following steps highlight the proposed Multi-View Object Detection pipeline:

- **Pre-processing** of the 3D object/scene (format conversion, scaling, centering, default orientation).
- **Rendering** the 3D object/scene to get the multi-view images for a set of parameters.
- **Inference** using the 2D object detection network for each one of the images.
- **Aggregation** of all the predictions into an annotation list.

Technical Details

The pre-processing procedure starts with the format conversion. 3D data come in many different formats; therefore, our pipeline should be able to process at least the most common of them. Since PyTorch3D[16] is our main renderer, which supports only obj and ply formats, we also employ the Trimesh library[17] for converting different formats to obj. Currently, the deployed solution supports the obj, glb, gltf, ply, off, and stl extensions. Following the format conversion process, the 3D object/scene is passed through the renderer for generating the multi-view images. The renderer depends on a series of parameters as presented below:

- **image_size:** The size of the rendered 2D output image. The smaller the size, the more pixelated the image will appear.

[15] Formally, colourful images are considered 3D data (two spatial dimensions; with the third being the colour). However, the dimensionality here expresses the spatial dimensions only, hence, images are abusively referred to as 2D.
[16] https://pytorch3d.org/
[17] https://trimsh.org/

- **camera_dist:** The distance between the camera and the object.
- **elevation:** The angle between the vector that is formed from the object towards the camera and the horizontal plane y=0 (plane xz). It dictates the directions of the camera.
- **azim_angle** Let's say you have a vector from the object to the camera and you project it onto a horizontal plane y=0. The azimuth angle is then the angle between the projected vector and a reference vector at (0,0,1) on the reference plane (horizontal plane). In essence, it expresses the side (e.g., left side, right side, front view, back view, etc.) from which we are looking towards the object.

The following code snippet describes the exact parameters used for the multi-view approach as implemented in the context of MediaVerse.

```
{
"image_size": 1024,
"camera_dist": [2],
"elevation": [0],
"azim_angle": [0, 45, 90 135, 180, 225, 270, 315, 360]
}
```

The selection of the aforementioned rendering parameters relies on preliminary experiments conducted on downloaded objects/scenes from the web. It is within our priorities to develop a pipeline that can work with any object. To this end, each object/scene is normalised to abide within the unit sphere. Hence, the camera, whose distance is set at 2, provides a nice overview of the object/scene. Moreover, having an elevation angle equal to 0, simulates the camera placement of photos taken by human beings in the real world. Finally, the azimuth values of 0, 45, 90, 135, 180, 225, 270, 315, and 360 provide a spherical overview of the object/scene. We should note that a wide variety of parameter values have been tested throughout the preliminary experiments. However, the selected were the ones that systematically offered the most satisfactory results.

Finally, each of the images is passed to the object detection model that is already deployed in the annotation API (see D3.1 - Next Generation Content Model and Algorithms for New Media Types[18], for more details). The final annotation list contains the unique aggregated predictions of each multi-view image.

2.3.3 Results of Multi-View Object Detection

In this sub-section, we present some of the results using the multi-view pipeline for 3D annotation (Table 8 and 9). In the absence of a suitable dataset, the results are presented in terms of visual (qualitative) inspection.

Table 8: Example of 3D object annotation.

EXAMPLE 3D	
Scenario	User queries a 3D object to the 3D annotation service
Request payload	

[18] https://mediaverse-project.eu/wp-content/uploads/2021/10/MediaVerse_D3.1_NextGeneration-ContentModel-and-Algorithms-for-NewMediaTypes.pdf

Response	success: 1
	detections {
	object: 'person'
	}

Table 9: Example of 3D scene annotation.

EXAMPLE 3D	
Scenario	User queries a 3D scene to the 3D annotation service
Request payload	

Response	success: 1
	detections {
	object: 'potted plant', 'couch', 'chair', 'bed'}
	}

The results demonstrate that the model correctly identifies the objects in both cases (single model and multi-model scenes respectively). We note that in the case of the multi-model scene, all identified concepts are part of the scene, apart from the 'bed' that has been erroneously identified as an object.

2.3.4 Limitations and Next Steps

The employed multi-view approach is only an initial approach towards a reliable annotation tool for 3D objects and scenes. Despite the encouraging results, it faces some limitations that should be addressed. The first relates to some particular scenes that depict indoor layouts. Such scenes may be enclosed within surroundings (e.g., walls). Hence, a normalised camera distance with a value of 2 would provide multi-view images completely outside the scene. Therefore, the rendered image will lack information and the annotation tool will not be able to detect any object. Hence, it is within our scope to establish a pipeline that will also examine the scene at different distance values. The second limitation is in line with the fact that the renderer has a strong impact on the resolution of the 3D object/scene. Similar to low image resolution, when the number of vertices in the mesh is low, the rendered multi-view images lack quality and therefore, the detector hardly annotates them correctly. Figure 9 shows that the quality of the mesh is low and, therefore, the detector cannot provide meaningful results.

Figure 9: An example of S3DIS dataset after mesh reconstruction. The low-quality rendering becomes evident.

Our priority is to face the described limitations and provide a more robust renderer for the 3D annotation task. Additionally, when it comes to direct 3D processing, the number of supported concepts is very limited due to the lack of a suitable dataset. Therefore, we are planning to extend our research to zero/few-shot learning (Zhao et al., 2021; Sharma & Kaul, 2020), to handle a wider spectrum of concepts through 3D processing.

2.4 Non-AI-Expert Model Building Service

Task 3.1 will not only provide powerful pre-trained models for automatic media annotation but also will enable non-AI-experts to build their own models for niche tasks pertinent to their work, through a user-friendly training module oriented to amateurs.

2.4.1 Existing Related Services

Before beginning to build our own user-friendly training module for MediaVerse, we start by investigating existing services and their features to maintain elements that serve the purpose of our module and discard others that defeat it.

Model Building

After extensive Google search, the following services for model building and training are identified as related to our work and are elaborated in Table 10.

Table 10: Existing model building services.

DeepVA, https://deepva.ai/	
It provides pre-trained models for face, logo, landmark recognition, and many other tasks. It also enables users to train their own models and expand upon the existing ones. Training data acquisition is said to be facilitated through automatic procedures. Finally, one appealing feature is that they claim that models can be built with only one image (or very few) through few shot learning techniques.	**pros:** good interface, nice graphics, easy to use **cons:** it's a black box, closed-source solution
Create ML, https://developer.apple.com/documentation/createml	
This is a tool for Apple devices and operating systems like Swift and macOS playgrounds and is used to create and train custom machine learning models. It enables one to train models to perform tasks like recognizing images, extracting meaning from text, or finding relationships between numerical values. It provides a friendly environment for data ingestion and model training without the need of user defined architectural details.	**pros:** nice instructions page, it is as high level as it should for a non-AI-expert **cons:** The tasks provided as options to the user could be even more abstract.
Deep Learning Studio, https://deepcognition.ai/deep-learning-studio/	
It is free to use and can run locally or in the cloud. It provides developers with pre-configured environments. The GUI development platform makes building models efficient without coding. It supports Chainer, H20, Keras, Mxnet, Tensorflow, Caffe, Chainer, Pytorch. All models are saved as experiments and can be compared to previous work. You can switch back to old models if needed. To	**pros:** it supports all deep learning elements a non-expert needs to build a model, from data ingestion through model building, tuning and deployment. **cons:** (1) data preparation is not so easy (certain naming conventions, csv file of images required by the user, no randomization supported), (2) not too high level (one

assist you with model optimization, tabular comparisons of all your model changes are available. You can upload data, it supports pre-trained models, it provides multiple GPUs for timely hyperparameter optimization and easy model deployment through REST API service.	needs to know what flatten, dense, dropout, etc layers are and how they are used; you need to define the architecture and hyperparameter values, you just don't code it)
Tensorflow's Deep Playground, https://playground.tensorflow.org/	
It lets one to play with four simple datasets and construct simple feed-forward models to perform 2-class classification. Enables tweak of input features, number of units, learning rate, batch size, activation function, regularisation, data noise and split. It says it can be repurposed using the open GitHub repo, under Apache licence.	**pros:** visualisation features **cons:** (1) only for educational purposes, too narrow in terms of datasets and model architectures, (2) does not present accuracy, only loss which is less intuitive for non-experts
ANNdotNET, https://github.com/bhrnjica/anndotnet	
A deep learning GUI that lets you build and train neural networks for many different classification and regression tasks.	**pros:** supports many features to build a good model **cons:** it is low level in terms of machine learning so not oriented to non-AI-experts, you just don't need to code the model and process
NVIDIA DIGITS, https://developer.nvidia.com/digits	
It supports training of deep neural networks for image classification, segmentation, and object detection tasks. Also, it provides user-friendly interface for common deep learning procedures such as managing data, designing, and training neural networks on multi-GPU systems, monitoring performance in real time with advanced visualisations, and selecting the best performing model from the results browser for deployment. It is interactive so that the user can focus on designing and training networks rather than programming and debugging.	**pros:** many useful features for deep learning experts **cons:** it is for data scientists
MATLAB's Deep Network Designer, https://www.mathworks.com/videos/interactively-build-visualize-and-edit-deep-learning-networks-1547156558295.html	
It is a point-and-click tool for creating and modifying deep neural networks.	**pros:** almost anything could be done by coding **cons:** it is still low level, one needs to know how to connect different layers, hyperparameter values etc. to build a model. Also, after that the training is not done through GUI you need to code that.
ENNUI, https://math.mit.edu/ennui/	
It is a graphical tool for building neural network architectures and training them. It provides all standard layers (conv, dense, flatten, dropout, etc.) and activations (relu, sigmoid). You can export your trained model to python or julia.	**pros:** easy to use, but needs deep learning knowledge **cons:** (1) needs expertise on model building, (2) does not support other datasets other than MNIST
Azure Machine Learning, https://azure.microsoft.com/en-us/services/machine-learning/	
It enables users to build, deploy, and manage high-quality business-oriented machine learning models.	**pros:** includes autoML feature **cons:** oriented to experts

Data Labelling

In addition, annotation tools are partially related to the MediaVerse non-AI-expert model building service. The following annotation tools have been identified as related to our work:

- Make Sense - https://www.makesense.ai/
- CVAT - https://openvinotoolkit.github.io/cvat/about/
- Label Studio - https://github.com/heartexlabs/label-studio
- doccano - https://github.com/doccano/doccano
- prodigy - https://prodi.gy/

2.4.2 Discussion

The existing user-friendly deep learning services provide the convenience of data ingestion, preparation, pre-processing, model building, training, and evaluation through graphical, easy to follow and use interfaces. However, almost all services target data scientists, engineers and machine learning experts that want to build models without the burden of coding. This means that the end user needs to be aware of all jargon (e.g., layers, activation functions, losses, hyperparameters, etc.) and be knowledgeable in terms of model construction process (e.g., how the layers should be connected).

This should not be MediaVerse's target, rather people from other domains, like news media, might want to build their own models and we could facilitate them with a high-level service. Only, DeepVA and Create ML are more abstract and require less details by the user which is closer to our idea of MediaVerse's deep learning service. Nevertheless, they could be even more abstract for instance, when the user selects the task to be addressed instead of image classification, action classification, and object detection that may seem as jargon we could adopt a more descriptive route with questions to the user like:

- What would you like to do?
- Detect a person or multiple persons in an image?
- Recognize human activity in videos?
- Categorise the content of images?

Then the service could select the most suitable task and model automatically. Finally, in terms of graphics and analytics provided to the user, the directions of DeepVA and Tensorflow's Deep Playground seem appealing.

2.4.3 Related Work

Amateur users are not expected to put a lot of effort on data collection so we will have to provide a model building procedure that addresses the small training dataset issue. The most suitable direction for model building when only few data points are available is few-shot learning. Many studies have adopted the meta-learning or episodic training scheme to address few-shot scenarios. Learning class differences through few-shot training episodes on other tasks and then using a support set to obtain the task dependent information to evaluate the method on a test set is a typical approach. Other few-shot approaches exist involving data augmentation, and algorithm optimisation. Additionally, other options are available. For instance, Kolesnikov et al. (2020) find that transfer learning from a large enough model pre-trained on a large and diverse enough dataset results in great few-shot performance on widely used benchmarks. Moreover, Chen et al. (2021) show that standard pre-training as well as a combination of standard pre-training and meta-learning result in comparable to state-of-the-art performance in few-shot scenarios. Finally, Chen et al. (2018) and Nakamura and Harada (2019) show that standard transfer learning can compete with state-of-the-art meta-learning algorithms in the few-shot setting.

2.4.4 Few-Shot Face Recognition

Inspired by the simple yet effective Prototypical Networks methodology and the Classifier-Baseline as proposed by Chen et al. (2021), we experiment with few-shot face recognition using a widely used pre-trained back-bone architecture, namely the VGGFace2 (Cao et al. 2018). This model is a convolutional neural network pre-trained on a large corpus of image data (~3.3M) with the objective of face identification for 8631 identities and thus can potentially provide useful face descriptors of any identity. More precisely, we consider as a face descriptor of an input face image the vector extracted from the layer adjacent to the classifier layer. This leads to a 2048-dimensional descriptor, which is then L2 normalised. To assess the model's ability to provide useful face descriptors of new identities, for evaluation we consider a different dataset than the one it was trained on, namely the CelebA dataset (Liu et al. 2015). CelebA contains 202,599 images with faces of 10,177 identities. Figure 10 illustrates the distribution of images per identity.[19]

The use case scenario is face verification, namely the user provides the system with k different images of an identity's face (support set) and given a new image with a face (query) the system outputs whether it is of the same identity or not. To simulate this scenario based on the CelebA dataset, we randomly sample k+1 images of a randomly sampled identity considering the first k as support set and the last 1 as query. We do that n=1000 times. Additionally, we randomly sample k+1 images of two randomly sampled identities: the first k from the first and the last 1 from the second. Again, we do that n=1000 times. In that way, we obtain 2000 random sets of k+1 samples to perform our experiment. Half of them concern the same identity and the other half concern a different identity.

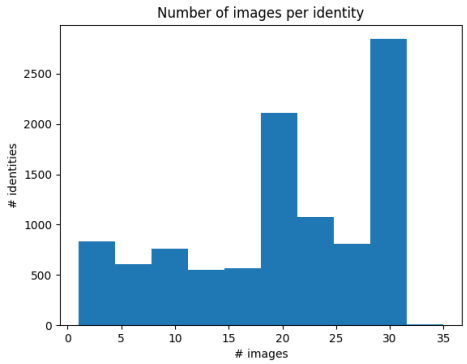


Figure 10: Number of images per identity histogram.

All 2000 * (k+1) images are provided as input to VGGFace2 which outputs the corresponding embeddings. The average embedding across the k support samples is considered as each identity's prototype and then we compute Euclidean distance and cosine similarity between the prototype and the query for all examples. We expect the distribution of distances among images of the same identity to have smaller values than the distribution of distances among images of different identities (opposite for similarity) and to be easy to discriminate between the two distributions using an appropriate threshold. To find the most suitable threshold we compute binary classification accuracy (same vs. different identity) for each candidate threshold and select the best. The candidate thresholds for Euclidean distance derive from the largest possible range [0,2] (note that the image embeddings are L2 normalised) as well as for cosine similarity being [-1,1].

[19] Unfortunately, CelebA does not provide the identity names to check for overlaps with VGGFace2 train split. Hence, we additionally performed the same experiment using the VGGFace2 test split that contains 500 distinct identities. The results are very similar in both experiments, so we only present the results from the CelebA dataset here.

Table 11: Binary classification performance for each k

	EUCLIDEAN DISTANCE		COSINE SIMILARITY	
k	**Accuracy**	**Threshold**	**Accuracy**	**Threshold**
1	88.2%	1.09	88.1%	0.405
2	90.2%	0.97	90.3%	0.465
3	93.9%	0.925	93.6%	0.495
4	92.9%	0.92	92.9%	0.490
5	93.7%	0.885	93.5%	0.530
6	94.0%	0.87	93.6%	0.525
7	94.3%	0.875	93.9%	0.545
8	95.1%	0.88	95.0%	0.540
9	94.1%	0.89	94.0%	0.510
10	94.8%	0.86	94.8%	0.540
11	94.3%	0.885	94.0%	0.540
12	94.8%	0.845	94.6%	0.565
13	95.5%	0.855	95.3%	0.555
14	94.3%	0.86	94.3%	0.555
15	93.8%	0.825	93.8%	0.585
16	94.8%	0.85	94.5%	0.560
17	95.5%	0.86	95.4%	0.525
18	95.2%	0.825	95.1%	0.585
19	95.2%	0.855	95.1%	0.550
20	94.7%	0.855	94.3%	0.550

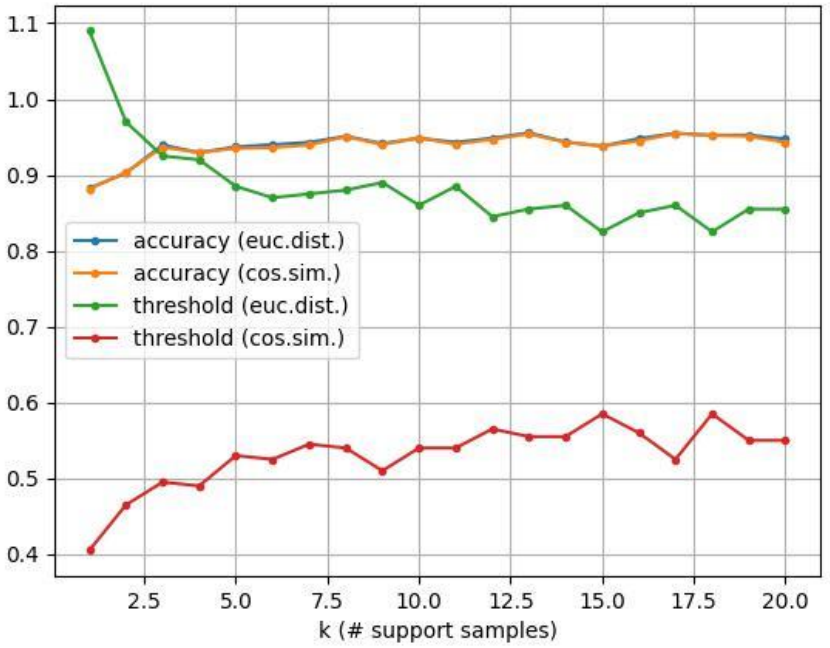


Figure 11: Binary classification performance for each k.

In Table 11 and Figure 11, we present the corresponding results for different support set sizes k=1,...,20. It seems that the performance is not bad even in the case of one-shot face recognition, which results in 88.2% accuracy when using Euclidean distance and 88.1% when using cosine similarity. Also, as the number of support examples increases the accuracy is increased as well, reaching 95.5% when k=13 and then remains around that plateau. The appropriate threshold is not flat over different k, rather it decreases with k for the Euclidean distance case. This is an expected effect as distances among instances of the same identity are expected to get smaller when averaging across a cloud of points to compute the identity's prototype. Similarly, as similarity is the opposite of distance, it is expected that the threshold in the case of cosine similarity increases with k. In Figures 12, 13 and 14, we present the distance and similarity distributions of same and different identities for 1-shot, 5-shot and 10-shot face recognition, respectively.

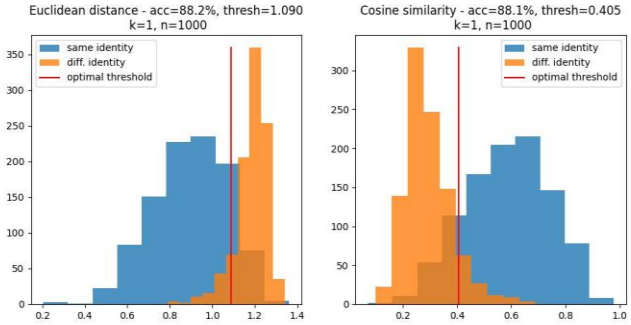


Figure 12: Histograms of distance and similarity for the same and different identity (k=1).

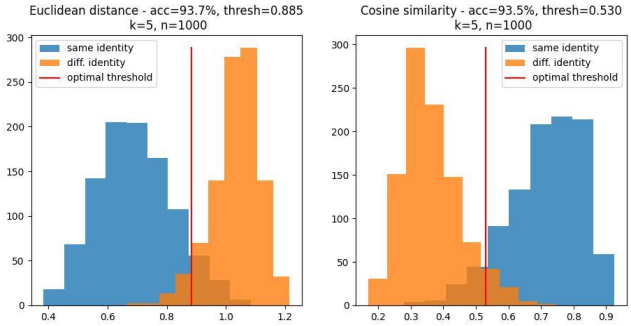


Figure 13: Histograms of distance and similarity for the same and different identity (k=5).

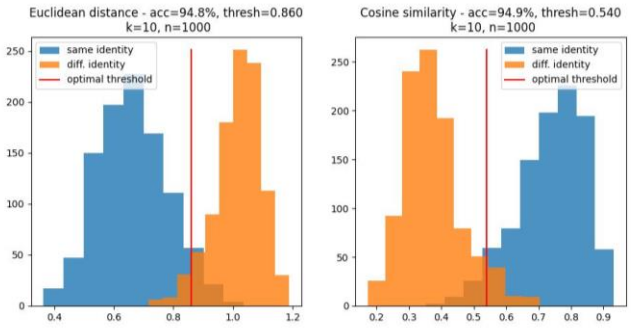


Figure 14: Histograms of distance and similarity for the same and different identity (k=10).

2.4.5 Few-Shot Image Classification

Recently, many works argued that typical fine-tuning of pre-trained models can lead to great few-shot performance and sometimes even better than using the standard meta-learning techniques (Kolesnikov et al. 2020; Chen et al. 2021; Chen et al. 2019; Nakamura and Harada 2019). The critical point is that the upstream training dataset must be diverse and big enough (a lot bigger than ImageNet-1k) to let deep models gain great transferable knowledge.

We replicate experiments from BiT paper[20] but also using other pre-trained models and exploring the hyperparameter space to obtain optimised few-shot performance in four datasets (i.e., CIFAR10, MNIST, Fashion-MNIST, and 400-bird-species)[21]. More precisely, we consider the timm[22] implementations of ResNet18 pre-trained on ImageNet-1k as a baseline (R18), ViT small pre-trained on ImageNet-21k (ViT) and ResNetV2 50x3 pre-trained on ImageNet-21k (BiT) as well as a small parameter grid:

- learning rate: {0.01, 0.003}
- batch size: {32, 128}
- number of shots (k): {5, 10, 50}
- number of training iterations: {500, 1000}

Additionally, for each experimental configuration we consider three repetitions using different random seeds to obtain performance variation estimations. In total, we conducted 864 reproducible experiments and the corresponding code is available in our GitHub repository[23].

As a pre-processing step on the training data, we consider grey-scale expansion from 1 to 3 dimensions (if applicable), resize, random crop, random horizontal flip, and standardisation per channel. Then, for each experiment, we randomly sample k images per class and form the training batches[24]. For the evaluation, we consider the whole test splits, and we apply grey-scale expansion, if applicable, resize and standardisation per channel. Unfreezing the whole model or a part of it did not bring any significant gains so we keep the pre-trained models frozen and only train the classification layers. During training, we reduce the learning rate by a factor of 10 at 30%, 60% and 90% of the iterations. We consider the categorical cross-entropy loss function and the stochastic gradient descent optimizer. The training is conducted on a GTX1080 for CIFAR10, MNIST, Fashion-MNIST and on a RTX3060 for 400-bird-species. Finally, we evaluate the models' performance using the standard accuracy metric. In Figure 15, we present the results in terms of performance and time for all hyperparameter values. Each blue box plot represents the accuracy distribution, and each red box plot represents the time distribution, over all experiments with a certain hyperparameter value (e.g., when training ResNet18 or when k=5 shots are used per class). We observe high accuracy scores in all datasets, and more precisely ViT outperforms R18 and BiT. BiT yields good results in many cases as well, however, it is a lot slower to train. R18 is very fast to train but it exhibits poor performance as expected.

[20] https://arxiv.org/abs/1912.11370
[21] https://www.kaggle.com/gpiosenka/100-bird-species
[22] https://github.com/rwightman/pytorch-image-models
[23] https://github.com/ckoutlis/mv-model-building-gui
[24] If the size of the few-shot training dataset is smaller than the batch size we resample with replacement.

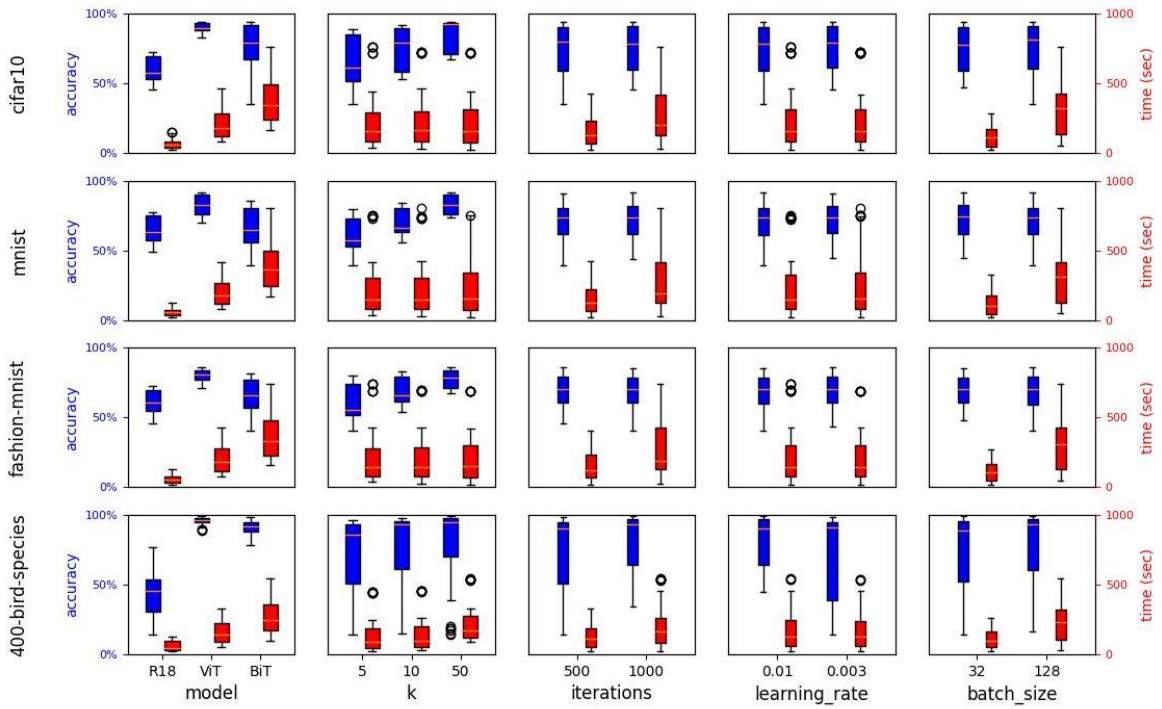


Figure 15: Distribution of accuracy (blue) and duration (red) for each hyperparameter value and dataset.

The main reasons for that are (a) its low capacity and (b) the upstream training dataset's size. Hence, ViT seems to provide the best trade-off between training time and performance. Also, the accuracy increases with k, which is an anticipated effect while time is not significantly affected. The number of iterations although it affects time, it does not seem to affect performance. Learning rate provides similar results in both cases and similar training times. Finally, batch size results in similar accuracy in both cases but time is significantly affected when a larger value is selected. Similar findings are encountered in the analysis of all datasets.

To select the hyperparameters of the training process that will support the module for image classification, performance is equally important with time. So, we do not only search for the configuration that yields maximum accuracy rather we heuristically search for the most efficient configuration. For each k, we consider all conducted experiments and the corresponding (time, accuracy) tuples. First, we identify the experiment with maximum accuracy α, then we filter out all experiments with accuracy less than α-5% and finally from the remaining we opt for the experiment that needs the minimum time and call it 'efficient'. In Figure 16, we present the results of this procedure for all datasets. In most cases, significant gains are obtained in terms of time with minimum losses in terms of performance. Table 12 presents the most efficient configuration for each k and dataset.

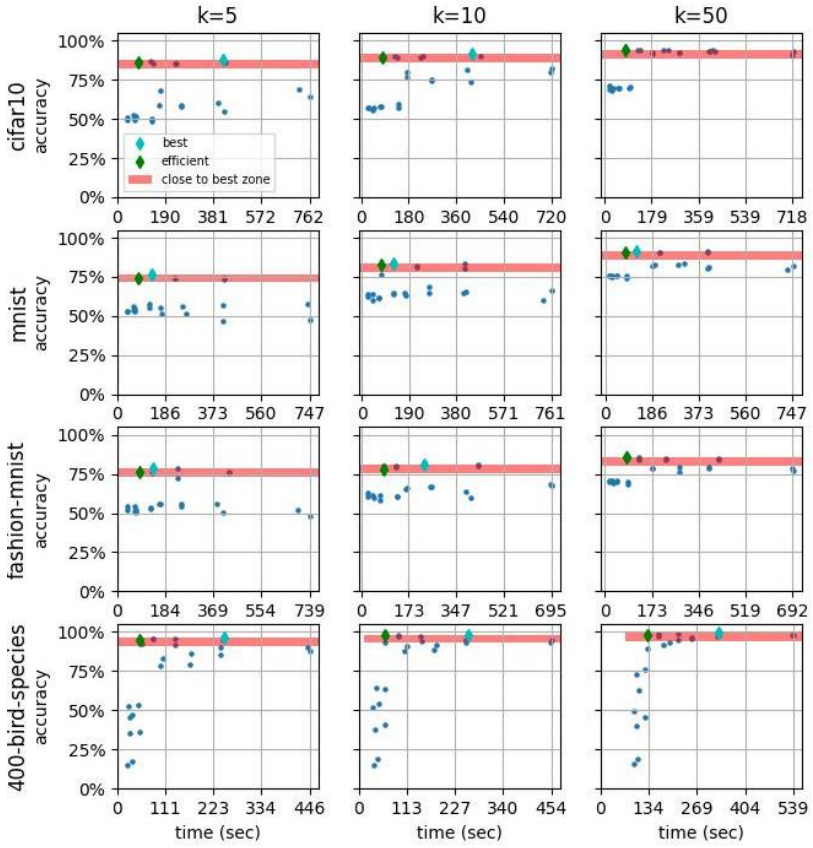


Figure 16: Heuristic approach for identification of the most efficient experimental configuration.

Table 12: The most efficient experimental configurations per k and dataset

	K		
	5	10	50
CIFAR10			
model	ViT	ViT	ViT
learning rate	0.003	0.003	0.003
batch size	32	32	32
iterations	500	500	500
accuracy	86.32%	89.70%	93.91%
time	82.8 sec	80.7 sec	80.5 sec
MNIST			
model	ViT	ViT	ViT
learning rate	0.01	0.003	0.01
batch size	32	32	32
iterations	500	500	500
accuracy	74.43%	82.57%	90.56%
time	81.7 sec	79.5 sec	78.4 sec

Fashion-MNIST			
model	ViT	ViT	ViT
learning rate	0.01	0.01	0.003
batch size	32	32	32
iterations	500	500	500
accuracy	76.68%	77.91%	85.39%
time	83.5 sec	81.1 sec	80.3 sec
400-bird-species			
model	ViT	ViT	ViT
learning rate	0.01	0.01	0.01
batch size	32	32	32
iterations	500	500	500
accuracy	95.22%	97.67%	97.73%
time	52.7 sec	61.8 sec	129.6 sec

Apparently, there is huge overlap in configuration hyperparameters across different datasets and number of shots k. In all scenarios, ViT model with batch size 32 and 500 iterations provide the most efficient configuration. Learning rate in most cases is 0.01, and thus we opt for it, although in 5/12 of the cases it is 0.003. It is important to mention that reported times include data loading, model loading, model training and model evaluation and that for 400-bird species the times are in general smaller because the training was performed on a better GPU.

2.5 Deployment

The media annotation service exposes a gRPC API that the MediaVerse DAM consumes. In addition to the bidirectional streaming endpoint, discussed in D3.1 - Next Generation Content Model and Algorithms for New Media Types[25], we have implemented three unary RPC endpoints, ImageAnnotation, ThreeDAnnotation and VideoAnnotation, responsible for annotating images, 3D assets and videos respectively (Table 13). The motivation for these endpoints is that they are simpler to use, as the client does not need to implement a connection management scheme for handling the gRPC stream. In each case, the request message contains a URL to the media file that is to be analysed and the response message contains the results of running all the relevant to the media type annotation models.

Table 13: Unary RPC endpoints

```
syntax = "proto3";

service MVAnnotationServer {
  rpc ImageAnnotation(ImageAnnotationRequest) returns (ImageAnnotationResponse);
  rpc ThreeDAnnotation(ThreeDAnnotationRequest) returns (Annotation3dResponse);
  rpc VideoAnnotation(VideoAnnotationRequest) returns (VideoAnnotationResponse);
}

message ImageAnnotationRequest {
  string image_url = 1;
```

[25] https://mediaverse-project.eu/wp-content/uploads/2021/10/MediaVerse_D3.1_NextGeneration-ContentModel-and-Algorithms-for-NewMediaTypes.pdf

```
}
message ImageAnnotationResponse {
  int32 status = 1;
  string error_msg = 2;

  ImageActionRecognitionResponse image_action_recognition_result = 10;
  ImageCaptioningResponse image_captioning_result = 11;
  ImageCrossModalEmbeddingResponse image_cross_modal_embedding_result = 12;
  ImageDisturbingContentResponse image_disturbing_content_result = 13;
  ImageFaceRecognitionResponse image_face_recognition_result = 14;
  ImageMemeDetectionResponse image_meme_detection_result = 15;
  ImageObjectDetectionResponse image_object_detection_result = 16;
}

message ThreeDAnnotationRequest {
  string asset_url = 1;
  string asset_type = 2;
}
message ThreeDAnnotationResponse {
  int32 status = 1;
  string error_msg = 2;

  repeated string object_detections = 10;
}

message VideoAnnotationRequest {
  string video_url = 1;
}
message VideoAnnotationResponse {
  int32 status = 1;
  string error_msg = 2;

  VideoActionRecognitionResponse video_action_recognition_result = 10;
  VideoDisturbingContentResponse video_disturbing_content_result = 13;
  VideoFaceRecognitionResponse video_face_recognition_result = 11;
  VideoObjectDetectionResponse video_object_detection_result = 12;
}
```

As described in D3.1, the media annotation controller, shown above in Figure 17, implements the gRPC server that the clients connect to while the execution of the core models is delegated to an instance of the NVidia Triton Inference Server. That is, the controller accepts client requests and performs any pre-processing steps needed, then sends a gRPC request to the Triton server, post-processes the results and finally responds to the client. The only model that is not running on Triton is the 3D annotation one as many of the operations performed in 3D deep learning models have not yet been standardised and included in the ONNX opset. The 3D annotation model is thus implemented as a separate service based on Pytorch 3D, which the controller queries internally through a REST HTTP API.

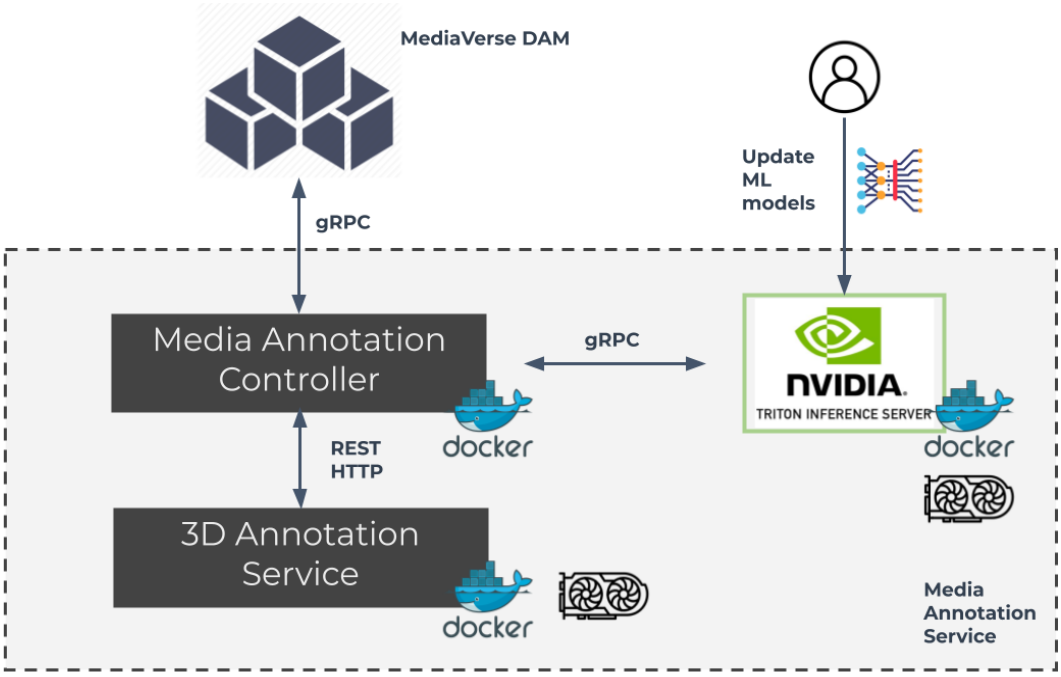


Figure 17: The Media Annotation Service architecture.

3 Cross-Modal Retrieval Technology

An Information Retrieval (IR) process begins when a user enters a query into the system. Queries are formal statements of information needs, for example, search strings in web search engines. In information retrieval a query does not uniquely identify a single object in the collection. Instead, several objects may match the query, perhaps with different degrees of relevancy. An object is an entity that is represented by information in a content collection or database. User queries are matched against all database objects; however, as opposed to classical SQL queries, in IR the results may or may not match the query exactly, so that they are typically ranked according to their relevance instead. This ranking of results is the key difference of information retrieval searching compared to database searching (Figure 18).

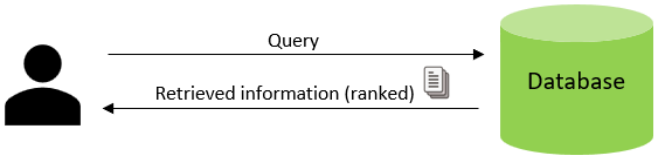


Figure 18: Information retrieval.

Typically, IR systems are designed to operate on one domain at the time (e.g., text vs text or image vs image), however, **cross-modal retrieval** aims to enable flexible retrieval experience across different modalities (e.g., texts vs. images). The core of cross modal retrieval research is to learn a common subspace where the items of different modalities could be directly compared to each other.

In this section, the focus is on content retrieval, also called "search", which requires a user to actively insert a query (a text or an image) as input. The user aim is to find the MediaVerse contents most similar to its input query regardless of it being a text or an image. Multimodal retrieval can be seen as a parallel process with respect to pure text-base one. It is necessitated because some assets do not have high quality text metadata provided by uploaders (because users do not provide that type of information), or annotation services in many cases suffer from a wrong labelling problem, leading to false positive results during retrieval. Hence, we need directly to process the content itself, extracting meaningful features from it. Extracting features directly from an image can provide us with more information than simply relying on image tags, such as the relative object position, their number or colour. Therefore, we are interested in how to extract efficiently information from data, which is modality agnostic, semantically relevant and not susceptible to errors due to previous annotations.

3.1 Related Work

To benefit from the abundance of multimedia data and make optimal use of the rapidly developing multimedia technology, we need automated mechanisms. These mechanisms establish a similarity link from one multimedia item to another one if they are semantically related, regardless of the type of modalities (e.g., text, visual or audio) of the items. Modelling of similarity links has traditionally focused mainly on single-modality scenarios. For instance, information retrieval has focused on bringing similar textual documents together, while content-based image, video or audio retrieval has attempted the same for images, videos, and audio items, respectively.

These areas of research dates to the mid-90s with the study of Mori et al. (1999) as a representative early work. Over time, research shifted towards learning joint multi-modal embedding spaces (Yang et al., 2016; Yang et al., 2014) with techniques like kernel Canonical Correlation Analysis, graph-based methods (Zhai et al, 2014; Zhuang et al, 2013) and various ranking objectives (Hodosh et al., 2013). Over time, work explored many combinations

of training objective, transfer, and more expressive models and steadily improved performance (Kiros et al., 2014; Faghri et al., 2017).

Since features of different modalities usually have inconsistent distribution and representation, a modality gap needs to be bridged. A common approach to bridge the modality gap is representation learning. The goal there is to find (i.e., learn) projections of data items from different modalities into a common (modality agnostic) feature representation subspace in which the similarity between them can be assessed directly. A variety of cross-modal retrieval methods (Feng et al., 2014; Hardoon et al., 2004; Peng et al., 2016; Wang et al., 2013) have been proposed recently, which propose different ways of learning the common representation subspace. Cross-modal retrieval (Wang et al., 2016) has attracted great research interest lately. (Chen et al., 2020) review the related literature and classify the models for this problem in four main categories, namely pairwise, adversarial, interaction and attributes learning models.

In general, we can also separate the related work in: 1) methods that use large-scale visual-language pre-training of huge BERT-based models (the current state of the art results in image-text retrieval); 2) methods focusing on methodological improvements on neural network architectures that are trained on the dataset of interest, for example the MSCOCO, Flickr8K or Flickr30K.

More precisely, Lu et al. (2019) extend the BERT model by adding also region features from an object detection model as input and a co-attention mechanism to introduce interactions in some layers. Chen et al. (2020) consider four pre-training tasks: masked language, masked region, image text matching and word region alignment. They mask one modality at a time unlike other previous studies and, they train with respect to one task per minibatch. Li et al. (2020) utilize a BERT model with triplet input (sequence and object labels, region representations from faster RCNN) and two combined loss functions: masked token loss and contrastive loss. Finally, Radford et al. (2021) and Jia et al. (2021) follow a similar approach considering simple text and image encoders, yet of huge capacity, and the same contrastive learning framework. The former is trained on 400M image text pairs, while the latter on 1.8B noisy image text pairs from the web. All the aforementioned models are pre-trained on large-scale datasets composed of millions or billions of pairs and are then fine-tuned and evaluated on downstream understanding and generation tasks, one of which is image-text retrieval. Regarding the second class of models, in initial attempts (Kiros et al., 2015) an encoder-decoder RNN network is trained to predict the surrounding sentences in a text document. In such a way, they create generic and useful representations for text sentences. Then they use these representations for cross-modal retrieval among other tasks, with non-state of the art results but very close to at the time of publication. More recently, a cross attention mechanism is proposed for cross modal retrieval (Lee et al., 2018). Also, Wei et al. (2020) and Zhang et al. (2020) introduce attention mechanisms that consider self- and cross dependencies between the two modalities with comparable to state-of-the-art results. The former utilizes multi-head attention mechanisms on top of BERT word features and image features extracted by object detection models. He et al. (2020) consider a LSTM text encoder and a LSTM image encoder with attention on the feature map of image patches (created by VGG-19), with the number of time steps for image encoding being a hyperparameter. Finally, a graph-based approach is presented (Diao et al., 2021) where self-attention with average vector query is employed for final feature vector computation on both modalities. Global (between the final feature vectors) and local (between each word feature and the corresponding context vector from region features) similarity vectors between image and text are extracted and the two modules, (1) similarity graph reasoning and (2) similarity attention filtration compute the final similarity vectors which pass through fully connected layers to compute similarity score between image and text.

3.2 Neural-based Architectures for Cross-modal Retrieval

Due to its rapid development, Deep Neural Networks (DNN) have increasingly been deployed in the cross-modal retrieval context, and then in particular to exploit nonlinear correlations when learning a common subspace (Andrew et al., 2013; Feng et al., 2014; Ngiam et al., 2011; Peng et al., 2016). In T3.2, we developed a deep learning model that is able to give a similarity score between a textual query and an image. This model is the core of the interlinking services developed in T3.2.

In the next subsections, we describe the models we decided to exploit along with training procedure and the data we use to develop them, and the results obtained over metrics chosen from the literature. We start with the Siamese Bi-directional LSTM an architecture, which relies on the content annotation tools developed in T3.1. This solution has some limits, which motivated us to search for another alternative, CLIP (Contrastive Language-Image Pre-Training), a neural network trained on a variety of (image, text) pairs resulting in a promiscuous embedding vector space where similar images and texts are near in terms of Euclidean Distance.

3.2.1 Siamese BiLSTM

The Siamese BiLSTM is the same as presented in D3.1 - Next Generation Content Model and Algorithms for New Media Types[26], Section 4.2. We replaced this architecture with one based on CLIP due to the limits listed below.

Limits

We foresee four major limitations of this approach:

1. **Scalability**: The similarity between two elements is computed by the FNN block, which will then have to run every time a similarity score is required. Therefore, for N elements in the database and M queries, the FNN will have to run O(M x N) times. Therefore, the feature extraction process is dependent on the couple of assets compared. It can become a problem as soon as we have millions/billions of objects in the database and hundreds/thousands of queries to run at the same time.
2. **Visual blindness**: The system is partially blind when dealing with visual content since it considers only tags of the objects contained in the image. Therefore, the model is not able to distinguish between two different images containing the same objects (order, context).
3. **Sequence focused**: BiLSTM focuses on sequences. The annotations for images are not ordered and any order introduces artificial information in the process, which can weaken the model retrieval capabilities.
4. **Annotation dependency**: Errors in the annotation are propagated in the ranking process, compromising retrieval performances.

3.2.2 CLIP

State-of-the-art (SOTA) computer vision systems are trained to predict a fixed set of predetermined object categories. This restricted form of supervision limits their generality and usability since additional labelled data is needed to specify any other visual concept. Learning directly from raw text about images is a promising alternative, which leverages a much broader source of supervision. The simple pre-training task of predicting which caption goes with which image is an efficient and scalable way to learn SOTA image representations from scratch (Radford et al., 2021).

[26] https://mediaverse-project.eu/wp-content/uploads/2021/10/MediaVerse_D3.1_NextGeneration-ContentModel-and-Algorithms-for-NewMediaTypes.pdf

The model transfers non-trivially to most tasks and is often competitive with a fully supervised baseline without the need for any dataset specific training. Zero-shot CLIP matches or outperforms all prior zero-shot results on Flickr30k and MSCOCO datasets. Zero-shot CLIP is also competitive with the current overall SOTA for the task of text retrieval on Flickr30k. CLIP was designed to mitigate a few major problems in the standard deep learning approach to computer vision:

Costly datasets: Deep learning needs numerus data, and vision models have traditionally been trained on manually labelled datasets that are expensive to construct and only provide supervision for a limited number of predetermined visual concepts. The ImageNet dataset, one of the largest efforts in this space, required over 25,000 workers to annotate 14 million images for 22,000 object categories. In contrast, CLIP learns from text–image pairs that are already publicly available on the Web. Reducing the need for expensive large, labelled datasets has been extensively studied by prior work, notably self-supervised learning (Doersch et al., 2015; Grill et al., 2020), contrastive methods (Oord et al., 2018; Chen et al., 2020), self-training approaches, and generative modelling (Donahue & Simonyan, 2019; Kingma et al., 2014).

Narrow: An ImageNet model is good at predicting the 1000 ImageNet categories, but that is all it can do "out of the box." If we wish to perform any other task, a practitioner needs to build a new dataset, add an output head, and fine-tune the model. In contrast, CLIP can be adapted to perform a wide variety of visual classification tasks without needing additional training examples. To apply CLIP to a new task, all we need to do is "tell" CLIP's text-encoder the names of the task's visual concepts, and it will output a linear classifier of CLIP's visual representations. The accuracy of this classifier is often competitive with fully supervised models.

Training Data

According to (Radford et al., 2021), the algorithm has been originally pre-trained using three existing datasets:

- MS-COCO (Lin et al., 2014).
- Visual Genome (Krishna et al., 2017).
- YFCC100M (Thomee et al., 2016).

While MS-COCO and Visual Genome (Radford et al., 2021) are high quality crowd-labelled datasets, they are small by modern standards with approximately 100,000 training photos each. By comparison, other computer vision systems are trained on up to 3.5 billion Instagram photos. YFCC100M, at 100 million photos, is a possible alternative, but the metadata for each image is sparse and of varying quality. After filtering to keep only images with natural language titles and/or descriptions in English, the dataset shrunk by a factor of 6 to only 15 million photos. This is approximately the same size as ImageNet.

A major motivation for natural language supervision is the large quantities of data of this form available publicly on the Internet. Since existing datasets do not adequately reflect this possibility, considering results only on them would underestimate the potential of this line of research. To address this, a new dataset has been constructed of 400 million (image, text) pairs form a variety of publicly available sources on the Internet. The resulting dataset, named WebImageText, has a similar total word count as the WebText dataset used to train GPT-2.

Architecture

Figure 19 shows CLIP architecture at high-level. CLIP pre-trains an image encoder and a text encoder to predict which images were paired with which texts in our dataset. It is trained using a Contrastive Loss (He et al., 2020) that takes as input paired text and image embeddings (their respective representations). It brings to sample dependence within each batch since the Contrastive CLIP loss compares each sample in the batch against all

other samples within the batch (computing a batch similarity matrix). This entails that unlike most loss functions, the samples in each CLIP batch are not independent. In fact, CLIP is trained using a large batch size of 32,768 samples.

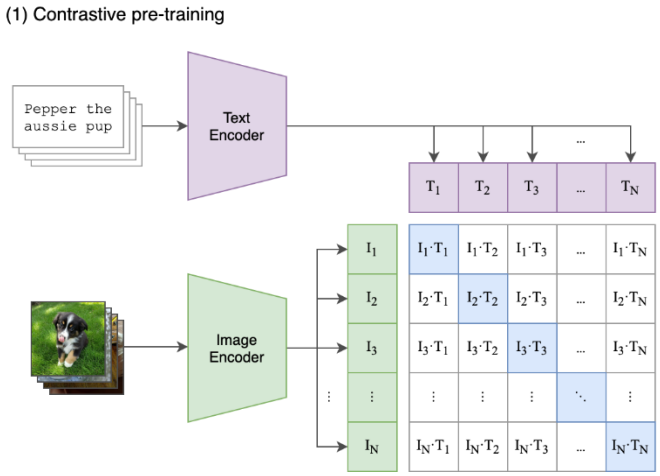


Figure 19: CLIP architecture.

Text Encoder: The text encoder is a Transformer (Vaswani et al., 2017). As a base size we use a 63M-parameter 12- layer 512-wide model with 8 attention heads. The transformer operates on a lower-cased byte pair encoding (BPE) representation of the text with a 49,152-vocab size. For computational efficiency, the max sequence length was capped at 76.

Vision Encoder: When pre-trained on large amounts of data, Vision Transformer (ViT) (Dosovitskiy et al., 2021) attains excellent results compared to state-of-the-art CNNs while requiring substantially fewer computational resources to train. The chosen configuration is **ViT-B/32** (ViT base variant with 32×32 input patch size).

Given the "Zero-Shot" nature of CLIP, we decided to load a pretrained version with a hyperparameter configuration specified the next sub-section.

Hyperparameters

Tables 14 and 15 show the hyperparameters adopted in (Radford et al, 2021). Adam has been used as optimization algorithm to minimize the Loss function. In the second table, there are the specifications of the two transformers (vision and text) together with the resolution of input images.

Table 14: CLIP general hyperparameters

HYPERPARAMETER	VALUE
Batch size	32768
Vocabulary size	49408
Training epochs	32
Maximum temperature	100.0
Weight decay	0.2
Warm-up iterations	2000
Adam β_1	0.9
Adam β_2	0.999 (ResNet), 0.98 (ViT)
Adam ϵ	10^{-8} (ResNet), 10^{-6} (ViT)

Table 15: CLIP encoders architecture.

MODEL	LEARNING RATE	EMBEDDING DIMENSION	INPUT RESOLUTION	VISION TRANSFORMER			TEXT TRANSFORMER		
				LAYERS	WIDTH	HEADS	LAYERS	WIDTH	HEADS
VIT-B/32	5×10^{-4}	512	224	12	768	12	12	512	8

3.3 System Architecture

This sub section describes the system architecture used to provide the retrieval service. As mentioned above, the retrieval task requires the user to insert a query. This query is firstly processed (encoded) and then compared to the other MediaVerse Contents previously stored. The user expects as output a ranked list of contents according to some predefined score. The system is composed of three main classes: CLIP encoder, the contents Container and the Faiss indexer (for similarity search)[27]. Each class was instantiated twice, for images and texts.

Container: By means of the Container, the system keeps track of which user posted which content along with the associated MediaVerse IDs. This structure is used because the recommendation system (section 4) must keep track of users inside the platform.

Table 16: Dictionary-like structure of the container.

```
Container = {
        Username : {
                Faiss_idx : [content_id, content_id, ...],
                }
        }
```

The container is structured as in Table 16, with a dictionary-like architecture, in which the main key is the **username**. The value contains another dictionary, whose keys are the Faiss indexes of the content. Each index contains a list of all the contents ids associated with that Faiss index. This means that if the same content is posted multiple times by the same user, there will be more content ids associated with the same Faiss index. In this way, the same content can be posted more than once without raising any error, but it will not be associated with a new Faiss index, thus saving space in the Faiss database. Another advantage of this structure is related to the fact that the main key is the username. It allows the recommendation system to retrieve the user's previous posts without searching the whole space. The DAM and the underlying mongoDB are used to store information about users.

[27] The container is a necessary structure because Faiss does not have a way of dynamically excluding vectors based on external criteria during the scanning of the space (faiss).

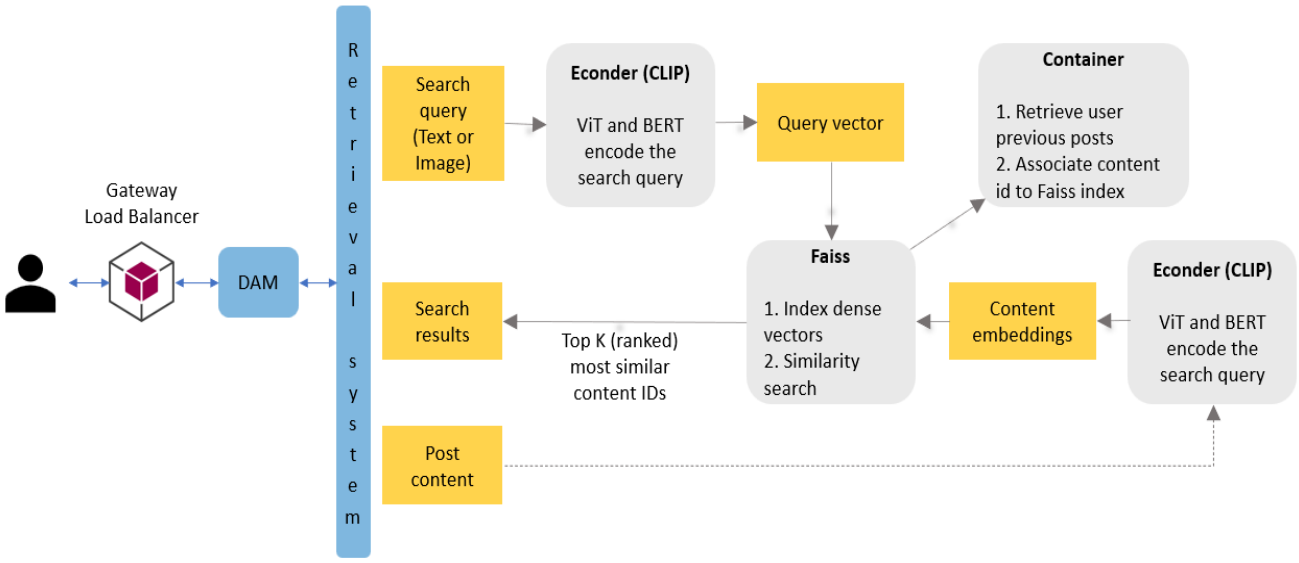


Figure 20: System architecture

The user can access the system (Figure 20) using two actions:

- **Post asset**: in this way the user can add an asset/content (text or image) to the system. It will be encoded to a 512 embedding vector and stored into the Faiss index.
- **Search query**: the user enters a query (text or image) with the aim of retrieving as a result the k elements most similar, to the submitted query, among those stored inside the Faiss index. The top K contents are ranked according to the cosine similarity with respect to the input query.

3.3.1 FAISS

Facebook AI Similarity Search[28] (Faiss) is one of the most popular implementations of efficient similarity search. It is a library — developed by Facebook AI — that enables efficient similarity search. So, given a set of vectors, we can index them using Faiss — then using another vector (the query vector), we search for the most similar vectors within the index. Faiss not only allows us to build an index and search — but it also speeds up search times to ludicrous performance levels. Faiss allows us to add multiple steps that can optimize our search using many different methods. A popular approach is to partition the index into Voronoi cells. Using this method, Faiss identifies a query vector x_q inside the cell it belongs to, and then use some similarity metrics to search between the query vector and all other vectors belonging to that specific cell. In that way, search produces an approximate answer, rather than an exact match as produced through exhaustive search.

Figure 21 shows the gains in search speed of Faiss with respect to different values of significant clusters (or cells) explored (nprobe).

[28] https://towardsdatascience.com/getting-started-with-faiss-93e19e887a0c

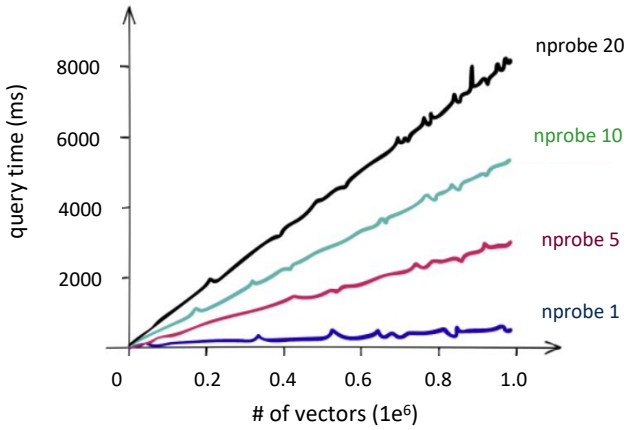


Figure 21: Faiss general performances. The query time for the index with different nprobe values.

Our Implementation

We adopted the **faiss.index_factory()** function, to instantiate the index. This function interprets a string to produce a composite Faiss index. The string is a comma-separated list of components. It is intended to facilitate the construction of index structures. We set the default value of nprobe=1 (search only inside the most significant cluster at retrieval time). The complete parametrized function is the following:

$$\text{Index} = \text{faiss.index\_factory}(512, \text{"Flat"}, \text{faiss.METRIC\_INNER\_PRODUCT})$$

In particular,

- input embeddings of size **512** are normalized (lie on the surface of a unit hypersphere) and added into the index as they are, without any encoding. It is accomplished with the **"Flat"** argument (it is equivalent to the IndexFlatL2 class).
- **"faiss.METRIC_INNER_PRODUCT"** has been chosen as metric to compute similarity between vectors. Since embeddings have been normalized, Inner product corresponds to **cosine similarity**.

Table 17 shows the run time (ms) needed to retrieve 5000 similar contents with respect to a query, comparing different alternatives.

Table 17: Search methods run time.

	N. CONTENTS	EMBEDDINGS SIZE	K	RUN TIME (MS)
vanilla cosine	5000	512	5000	129.31
filter + cosine	5000	512	5000	105.84
faiss	5000	512	5000	**0.72**

Among the different alternatives evaluated (vanilla cosine and filter plus cosine) Faiss considerably reduces the search time.

For each media type we instantiated one Faiss index. The motivation of this choice is discussed in section 3.4.2 in Experimental Setup in which the cosine similarity scores, obtained from CLIP vectors, are compared distinguishing between unimodal and cross-modal data. Figure 22 above shall express that every time we refer to the FAISS database, we refer to the two instances dedicated to text and image.

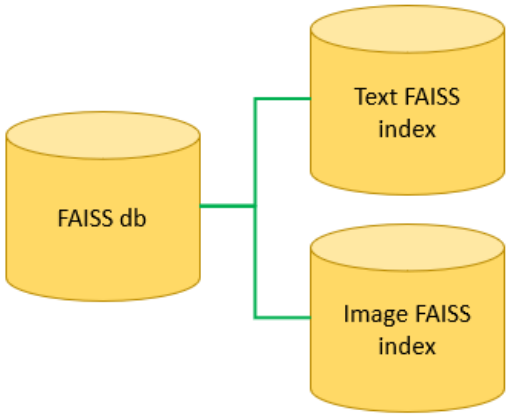


Figure 22: Faiss db split into two Faiss instances.

3.4 Experimental Setup

Pre-trained models: The pre-trained CLIP model is available online for inference and fine-tuning purposes. However, the publicly available versions do not include the best performing ones as reported in the paper. Thus, we opt for the best among the available ones, which is CLIP with ViT-B/32 vision encoder. We have downloaded and zero-shot evaluated this CLIP version on MSCOCO 5K test set giving R@1=0.29, R@5=0.54 and R@10=0.65 for image retrieval and R@1=0.48, R@5=0.73 and R@10=0.81 for text retrieval.

Other state of the art models like the Oscar pre-trained model, which has been fine-tuned on the downstream task of image-text retrieval (MSCOCO captions dataset) are available online for inference purposes. However, while the Oscar model outperforms CLIP on MSCOCO captions, the latter one is potentially a better choice as it has been trained on 400M image text pairs meaning that it recognizes many more than the MSCOCO concepts in visual and textual content and probably better fits MediaVerse's needs.

Multi-head cross-modal attention: We have also made attempts for methodological improvements of existing models in the same context, using transformer-like self-attention and cross-attention mechanisms on top of visual and textual encoders, either pretrained or trained from scratch together with the attention parts. Results on the MSCOCO 5K test set (Karpathy split) are presented in Table 18. It seems that with this setting we have not surpassed the state of the art but we are still exploring improvement options. However, we also came across a study (Wei et al, 2020) that implements a very similar approach, which provides state of the art results.

Table 18: Recalls of img2txt and txt2img tasks (MSCOCO val 5k)

	TEXT RETRIEVAL			IMAGE RETRIEVAL		
encoder	R@1	R@5	R@10	R@1	R@5	R@10
VGG19 BiGRU (3 layers, 128 units, 128 embedding dim)	0.094	0.283	0.406	0.086	0.260	0.379
CLIP (ViT-B/32)	0.322	0.623	0.747	0.237	0.518	0.656

Dataset: All the following results (3.5) are obtained by testing the whole system on **1000 samples of the 2014 validation split of MSCOCO**.

3.4.1 Similarity Comparison and Double Faiss Index

The objective of this study is to motivate the choice of instantiating two Faiss indexes, one for each modality (text and image), instead of a unique one. We start showing that CLIP embeddings are more likely to have higher similarity when they come from the same media type (e.g., text-text or image-image) rather than when they are from different media (e.g., text-image). This characteristic motivated us to create one Faiss index for each media type, so that, given a query, most similar results can be returned for each media type independently.

The following tables (Table 19, 20, and 21) show the similarity scores (cosine similarity) computed on the CLIP generated embeddings taken from the MSCOCO 2014, VIZWIZ (Gurari et al., 2020), FLICKER-8k validation splits (Hodosh et al., 2013). Each pair of modality combination (txt-txt, img_img, txt-img) were used to compute mean values of similarity scores among similar and dissimilar pair of contents. Blue tables show the cosine similarity values for pair of similar contents, red tables the value of the same metric for dissimilar contents.

There are no similarity values for similar image-image comparison since the mentioned datasets do not track similar images. Images were compared with each other only in terms of score among dissimilar images.

Table 19: MSCOCO - cosine similarity of similar and dissimilar content.

	SIMILAR CONTENTS		DISSIMILAR CONTENTS		
	Text-Text	**Text-Image**	**Text-Text**	**Image-Image**	**Text-Image**
Mean	0.823	0.299	0.590	0.501	0.157
Min	0.732	0.188	0.130	0.222	0.008
25° percentile	0.778	0.280	0.524	0.437	0.126
50° percentile	0.825	0.300	0.590	0.493	0.152
75° percentile	0.867	0.319	0.654	0.552	0.181
Max	0.914	0.397	1.0	0.895	0.340

Table 20: VIZWIZ - cosine similarity of similar and dissimilar contents

	SIMILAR CONTENTS		DISSIMILAR CONTENTS		
	Text-Text	**Text-Image**	**Text-Text**	**Image-Image**	**Text-Image**
Mean	0.772	0.289	0.622	0.588	0.199
Min	0.655	0.183	0.202	0.164	0.040
25° percentile	0.711	0.262	0.547	0.521	0.174
50° percentile	0.774	0.290	0.621	0.592	0.201
75° percentile	0.836	0.315	0.693	0.659	0.226
Max	0.895	0.392	1.0	0.956	0.351

Table 21: Flicker8k - cosine similarity of similar and dissimilar contents

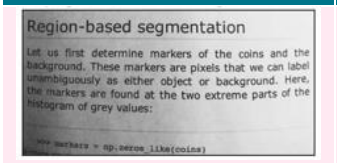
	Similar contents		Dissimilar contents	
	Text-Text	**Text-Image**	**Text-Text**	**Image-Image**	**Text-Image**
Mean	0.813	0.304	0.590	0.516	0.171
Min	0.720	0.217	0.198	0.171	0.002
25° percentile	0.766	0.285	0.528	0.450	0.144
50° percentile	0.816	0.304	0.595	0.511	0.170
75° percentile	0.859	0.325	0.658	0.576	0.197
Max	0.907	0.386	0.946	0.916	0.347

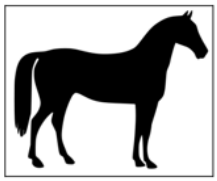
The tables above show that the similarity values differ in the case we do a unimodal (text-text) comparison with respect to the case of contents of different modality (text-image). By looking only at similar contents (blue), we can notice that scores associated to different modality are generally lower than a text compared to other texts. Interestingly, by looking at the dissimilar contents (red) inside the red box, we can see that if unimodal contents are not similar their scores are still higher than similar text-image. This results in dissimilar contents of the same data type with respect to the input query, to be prioritized with respect to contents of different type but semantically more similar. It gives rise to necessity of separating the two tasks to avoid unimodal contents to be prioritized over cross modal ones. Every observation made is consistent across the three datasets.

Example

Table 22 shows an example of the output obtained using one or two indexes is provided using 8 images (and their associated caption) randomly sampled from skimage (scikit-image is an image processing Python package that works with numpy arrays; the package is imported as skimage).

Table 22: Skimage images and their associated captions. Image ID and Text ID are the Mediaverse ID associated to the correspondent contents in the example provided for both single and double indexes.

Image	Image ID	Text	Text ID
	Z0WT7PJLGN	A page of text about segmentation	**QUERY** (queries are not present in the mediaverse network, so they are not associated with any id)
	1UPF6A04Z5	A portrait of an astronaut with the American flag	9YHU6WMZJE
	5R5YRSN2R0	A rocket standing on a launchpad	6QVH8CQKWFE

	T4H4YW70QQ	A facial photo of a tabby cat	XENG41O9Q2
	95O1N3NFXJ	A black-and-white silhouette of a horse	7Q1T2OOG10
	FN2JVSKW7N	A cup of coffee on a saucer	1ZAT41AU04
	1UPF6A04Z5	A person looking at a camera on a tripod	T6B41T0MSE
	IUNESE0KVX	A red motorcycle standing in a garage	061JYBF4NV

The caption “A page of text about segmentation” is used as search query, while all other contents (except the query) are inserted in the MediaVerse system through the Post content module. Thus, in this example, the Faiss index will store 8/8 images and 7/8 texts. Pay attention that only the contents posted in the MV network are associated with an ID, as opposed to the first (left out) text used as query instance. We can now distinguish between the two alternatives, single Faiss index or two separate Faiss indexes for texts and images.

Single Faiss Index

As we will explain later, the application gives as output a status code of the query operation together with a JSON dictionary containing two keys:

- contents: the ordered list of IDs recommended.
- scores: the cosine similarity scores associated to the IDs.

Figures 23 and 24 exemplify the shortcomings of instantiating a unique Faiss index for both texts and images.

QUERY: a page of text about segmentation
[RETRIEVE]-- status: 200. b'{"contents":["T6B41T0MSE","1ZAT41AU04","XENG41O9Q2","6QVH8CQKWF","9YHU6WMZJE","7Q1T2OO
G10","061JYBF4NV","Z0WT7PJLGN","95O1N3NFXJ","FN2JVSKW7N","1UPF6A04Z5","T4H4YW70QQ","IUNESE0KVX","5R5YRSN2R0","217F
NYO99P"],"scores":[0.6652600765228271,0.6554150581359863,0.619107723236084,0.6146003603935242,0.5892289280891418,0
.4901670515537262,0.48085999488830566,0.3587474226951599,0.22719718515872955,0.22674132883548737,0.225453257560729
98,0.22133639454841614,0.18296091258525848,0.17943187057971954,0.17070680856704712]}\n'

Figure 23: Single Faiss index output example.

We can notice that the first 7 IDs inside the red boxes are all text data - as verifiable by the table above – suggesting that if there is only one index for both text and images, the system tends to prioritize the contents of the same input modality. Similarity scores of dissimilar text data are higher than similar data of a different modality. In fact, the image associated to the input query (green box), with "Z0WT7PJLGN" as ID, is immediately after the last textual data, in the 8-th position. Hence, the system can correctly score (approximately 0.36) and retrieve the relevant image with respect to all other images, but after prioritizing all other texts.

Double Faiss Index

Given the limit emerged in the example above, it could be desirable to instantiate two separate Faiss indexes for texts and images. In this way, we aim to separate the tasks of cross-modal retrieval from the unimodal one at search time. The output below, in the case of two indexes, is slightly different: now there are two {contents:[], scores:[]} json elements wrapped into two keys, "image" and "text". In this way, the ranked list of contents returned is not mixed, but it is separated by data type.

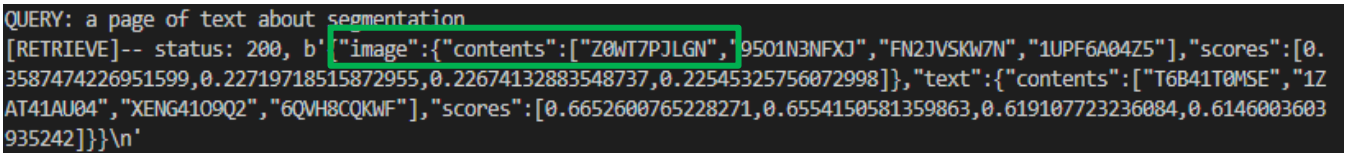


Figure 24: Double Faiss index output example.

The green box highlights that the relevant image is now prioritized in the correspondent images list. Irrelevant texts continue to appear in the texts list, but do not penalize relevant assets of different modality. Hence, we avoided to implement a single Faiss index in which insert all assets (texts and images), and we decided to instantiate two indexes:

- Text Faiss index where to add all text contents, and
- Image Faiss index where to add all image contents.

Separating indexes makes the retrieval unaffected by noise generated by unimodal contents (with respect to input query) with higher scores so, for example, given an input Text, its embedding is used as search key in both indexes retrieving the most similar k texts and images of their respective indexes.

3.5 Result and Discussion

Here we report our own experiments on performance evaluation of CLIP (ViT-B/32).

3.5.1 Recall

Firstly, the model has been evaluated in terms of recall which, in the content retrieval scenario, is equal to the fraction of the documents that are relevant to the query that are successfully retrieved:

$$recall = \frac{|\text{relevant documents}| \cap |\text{retrieved documents}|}{|\text{relevant documents}|}$$

Table 23 shows the values of recall for the tasks of Image Retrieval (text as input) and Text Retrieval (Image as input). R@k indicates the recall value when k elements are retrieved as recommend by the system. Hence, recall is a non-decreasing function of k. "avg. delta" stands for the percentage of recall lost, with respect to "vanilla cosine" on the entire collection, because of the approximated search (filter+cosine or Faiss).

Three scenarios have been evaluated:

- Vanilla cosine: the similarity search is made all over the content collection without any filtering.
- Filter+cosine: a filter is applied to reduce the cardinality of the pool. The cosine similarity is applied downstream with a smaller pool.
- Faiss: the similarity search is performed by means of the Faiss library (based on pool clustering).

Table 23: Recalls on MSCOCO 2014 val 1k. Comparison between different search techniques.

	IMAGE RETRIEVAL				TEXT RETRIEVAL			
	R@1	R@5	R@10	avg. delta	R@1	R@5	R@10	avg. delta
vanilla cosine	0.517	0.793	0.889	+0 %	0.548	0.825	0.910	+0 %
filter+ cosine	0.495	0.759	0.852	-4.235 %	0.541	0.814	0.897	-1.346 %
faiss	0.515	0.792	0.888	-0.208 %	0.547	0.824	0.909	-0.138 %

Values of Recall approach 0.9 in correspondence of 10 elements recommended by the system. In particular, the model gets systematically better recall values for Text Retrieval task than Image Retrieval one. Among filter+cosine and Faiss, the latter is proven to better preserve the recall while requiring less time to retrieve the contents. Substantially Faiss filter the pool in a better way, narrowing the search space only where there is high probability to find contents associated to higher similarity scores with respect to input query.

3.5.2 Mean Reciprocal Rank (MRR)

Another very insightful metric to evaluate content retrieval systems with, is **MRR** (Mean Reciprocal Rank). This not only considers the number of relevant contents returned by the system in the first K results, but also their position in the ranking. The reciprocal rank of a query response is the multiplicative inverse of the rank of the first correct answer: 1 for the first place, ½ for second place, ⅓ for third place and so on. The mean reciprocal rank is the average of the reciprocal ranks of results for a sample of queries Q:

$$MRR = \frac{1}{|Q|}\sum_{i=1}^{Q}\frac{\text{RelevanceLabelValue}}{rank_i}$$

where $rank_i$ refers to the rank position of the first relevant document for the i-th query.

Recall and MRR metrics have been used to evaluate the retrieval system on three tasks:

- text2image.
- image2text.
- text2text.

1000 samples are randomly taken from MSCOCO 2014 validation split to retrieve the results. Along the x-axis there is the number of retrieved contents k, from 1 to 100.

According to Figures 25 and 26, cross-modal retrieval (red & blue lines) achieves, on average, better performance compared with the uni-modal. The uni-modal (Text2Text) retrieval evaluation is obtained by considering positives pairs different captions associated to the same image. From our results it can be inferred that this is a harder task for CLIP to accomplish since metric values are generally lower than the other two curves. This is probably due to the fact that CLIP is optimized to the cross-modal scenario.

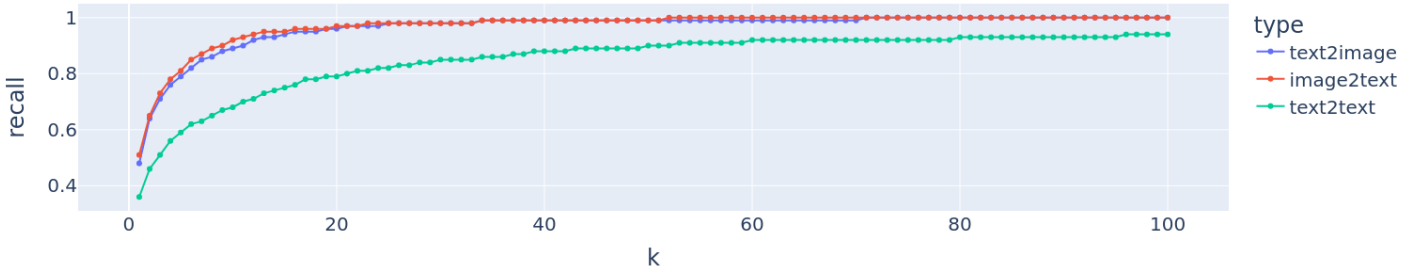


Figure 25: Recalls on MSCOCO 2014 val 1k with different number of retrieved contents (k).

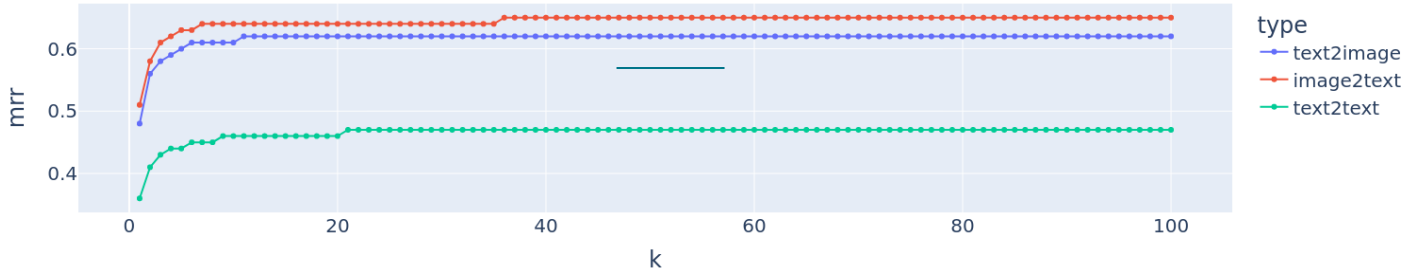


Figure 26: MRRs on MSCOCO 2014 val 1k with different number of retrieved contents (k).

3.6 Thresholding Analysis

In this sub-section, we discuss some efforts we did in the directions of improving the quality of results by returning a flexible number of results based on the relevance of the latter. In the previous sections, the user was required to insert a fixed number K of desired retrieved items. This static approach has the obvious drawback of returning irrelevant content when relevant items in the database are less than K, as well as omitting relevant content when the relevant items are more than K. It seems therefore reasonable to choose the number of items to be returned by the system based on their relevance to the query rather than by simply selecting the top K.

In this sense, different similarity thresholds have been tested to evaluate how many non-informative contents are filtered out (among the recommended ones) without losing retrieval Recall or MRR. Defined **t** the similarity threshold, and **score** the cosine similarity between input and content j:

- all assets associated to a **score** < **t,** are supposed to be less likely correlated with user input query.
- Conversely, if **score** >= **t**, we can confidently consider that asset linked to the input one.

It is a way to dynamically choose the number of assets retrieved by filtering out those less likely to be effectively similar to the input query. Hence, the input parameter K becomes a MAX meaning that it is the maximum number of retrieved items if all k output contents are considered similar to the input.

What is missing to apply this line of reasoning is the threshold value, **t**. As we have seen above, the cosine similarity varies if we consider unimodal or cross modal contents. With some extent it also varies if we compare img-img and txt-txt. Furthermore, the retrieval task seems to have slightly different performances among txt2img and img2txt, suggesting that different threshold should be defined task by task (txt2txt, img2img, txt2img, img2txt).

The following graphs show Recall and MRR values for different tasks. The objective is to understand a proper value of **t** such that we filter out as more contents as possible while retaining Recall and MRR retrieval performances.

On the y-axis there is the metric (Recall or MRR), along the x-axis "mean filter out" stands for the average number of contents filtered out when the threshold is **t** (higher value of t correspond to more filtered contents). The curve color encodes the number of k retrieved contents inserted by the user, showed values are 5, 10, 20, 50, 100, 150. The text label under each point encodes the value of **t** tested. Hence the graph has 4 measures:

- Recall: as defined in 3.5;
- K: MAX number of contents retrieved;
- t: similarity threshold;
- Mean filter out: mean number of filtered out contents out of k with threshold t.

It is called "mean filter out" since for each value of k, 1000 queries have have been performed each one resulting in a certain value of recall and number of filtered out items. Hence, both of them have been averaged over all 1000 query instances.

Threshold t, for the tasks of Text2Image and Image2Text retreieval, takes values: 0.20, 0.22, 0.23, 0.24, 0.245, 0.25, 0.26, 0.27, 0.28, 0.29, 0.30, 0.32. These numbers have been choosen referring to the similarity study of pairs text-image.

3.6.1 Text2Image

As regards, Text2Image the drop in recall (Figure 27) usually occurs for a threshold between 0.24 and 0.25 with up to abot 80% (on average) of k=150 contents filtered out. The same threshold is applicable preserving mrr which is more easily preserved until **t** = 0.26 (Figure 28).

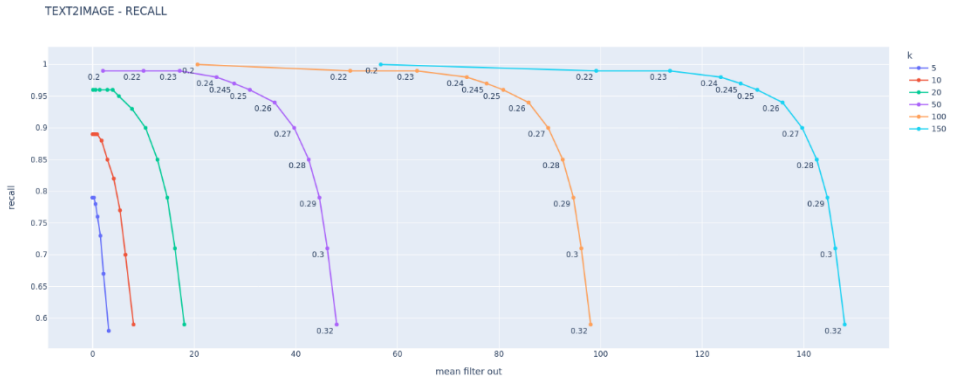


Figure 27: Text2Image recall thresholds.

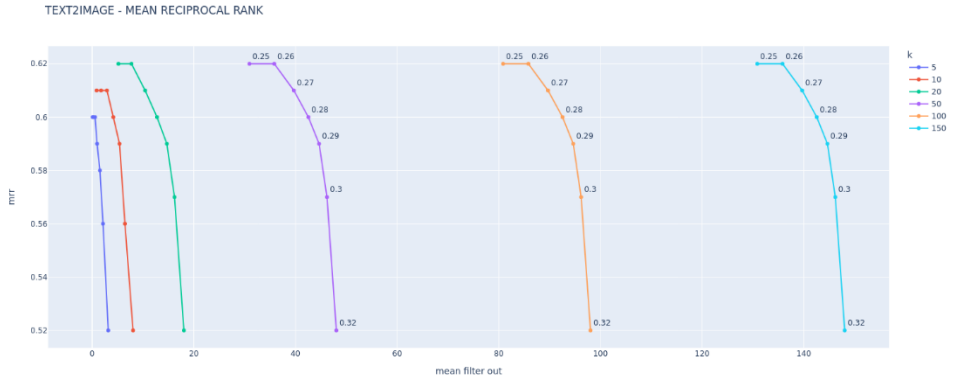


Figure 28: Text2Image mrr thresholdsImage2Text

For Image2Text the drop in recall (Figure 29) usually occurs for a threshold between 0.245 and 0.25 with up to abot 80% (on average) of k=150 contents filtered out. In this case mrr (Figure 30) drop occurs faster, with a **t** between 0.23 and 0.24. For this task a good threshold value could be 0.23, with still a good percentage of contents filtered out (75% for k=150).

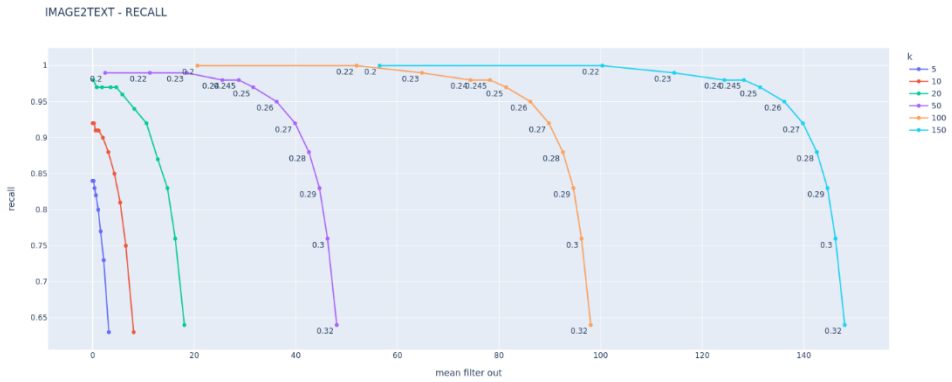


Figure 29: Image2Text recall thresholds.

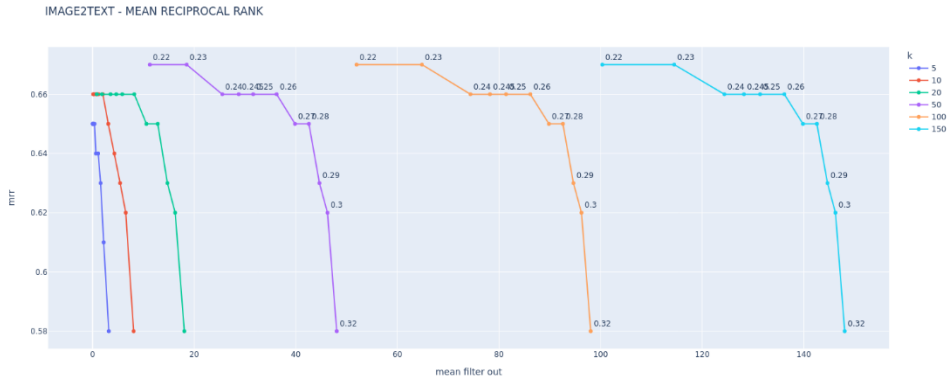


Figure 30: Image2Text mrr thresholds.

3.6.2 Text2Text

Different thresholds have been tested for the task of text2text retrieval given the similarity study in section 3.3. Here the tested values: 0.5, 0.6, 0.7, 0.74, 0.75, 0.76, 0.77, 0.78, 0.79, 0.80, 0.81, 0.82, 0.84, 0.86. For Text2Text the drop in recall (Figure 31) occurs very quickly in correspondence of a threshold about 0.6 with up to 25% (on average) of k=150 contents filtered out. In this case mrr (Figure 32) drop occurs in at the same value of **t**. For this task a good threshold value could be 0.23, with still a good percentage of contents filtered out (75% for k=150).

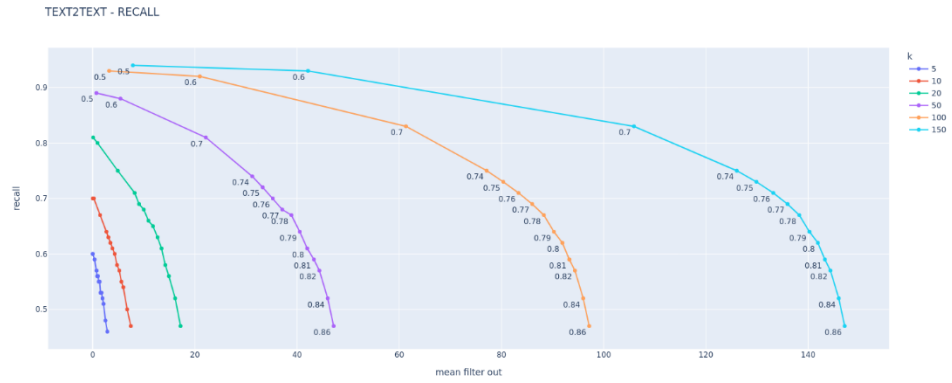


Figure 31: Text2Text recall thresholds

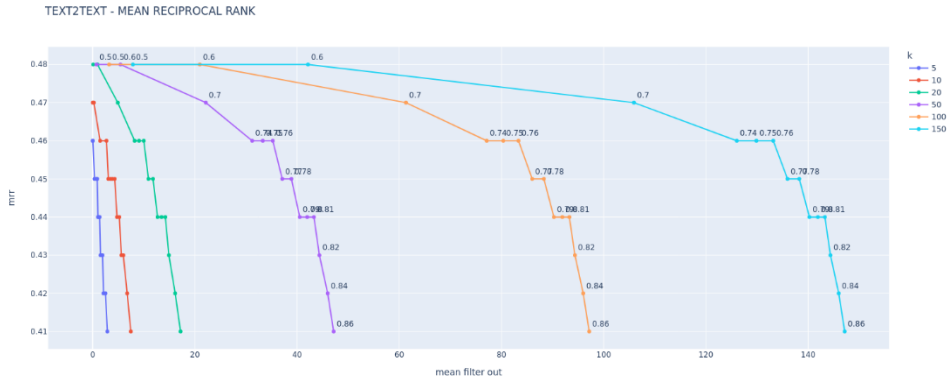


Figure 32: Text2Text MRR thresholds.

3.6.3 Image2Image

It was not possible to track similarities among images since the employed datasets do not aggregate images of the same semantics and context.

3.6.4 Conclusions

Since CLIP is a large-scale model, it is worth replicating these tests on different datasets as FLICKER8k and VIZWIZ to verify the robustness of threshold choice for each task.

3.7 Design and Deployment

The user can access the system by means of two functions (Figure 33):

- **add_content()**: in this way the user can add content (text or image) to the system. It will be encoded as a 512 embedding vector and stored into the Faiss index.
    - Input:
        - username: unique identifier of the user who posts the content;
        - text or image binary data (refer to usage.py example);
        - id: MediaVerse ID of the content to load;
        - type: "text" or "image", string describing the data type of the content to be loaded.
    - Output:
        - a success message in the field “msg”
        - the elapsed time for the operation in the field “time”

- **retrieve()**: the user enters a query (text or image) with the aim of retrieving the K most similar elements among those stored inside the Faiss index. These top K contents are ranked according to the cosine similarity with respect to the input query.
    - Input:
        - username: unique identifier of the user who search for a content;
        - text or image binary data (refer to usage.py example);
        - k: number of similar content to retrieve;
        - type: "text" or "image", string describing the data type of the input query.
    - Output:
        - retrieved texts in the field “text”:
            - “contents” is an ordered list containing the ids (string) of the retrieved texts. The list is ordered based on decreasing values of similarity scores (i.e., the first content is the best one retrieved (among all texts) for that query).
            - “scores” is an ordered list of similarity scores for the retrieved texts (i.e., the first score represents the similarity between the query and the first content in the “contents” field).
        - retrieved images in the field “image”:
            - “contents” is an ordered list containing the ids (string) of the retrieved images. The list is ordered based on decreasing values of similarity scores (i.e., the first content is the best one retrieved (among all images) for that query).
            - “scores” is an ordered list of similarity scores for the retrieved images (i.e., the first score represents the similarity between the query and the first content in the “contents” field).

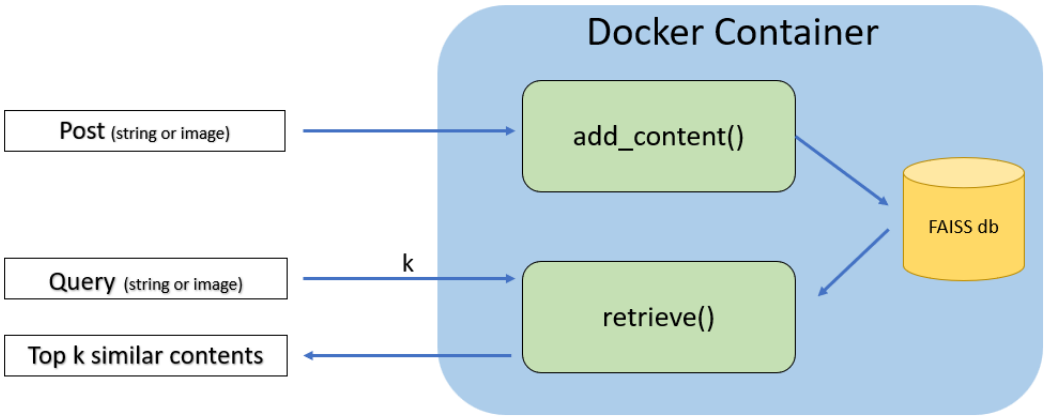


Figure 33: Rest API - Retrieval.

Duplicate content: The application is designed to check for duplicated contents even if they are associated to different MediaVerse ID. So, if the user tries to add an image/text already present in the collection but with a different ID, the application will append the new ID to the tail of the list of IDs associated to a specific Faiss index of the user. Hence, referring to the Container in section 3.3, the application is designed to not store multiple times the same content inside the Faiss index (for the sake of memory footprint), but it keeps track of different IDs associated to that same content. When the system is asked to retrieve the most similar contents with respect to an input query, the IDs associated to the retrieved contents are the most recent (last element of the IDs list).

3.7.1 Project Folder

The CMRR (cross-modal retrieval and recommendation) project is available on GitHub and contains (Figure 34):

- app.py: Flask Rest Api exposing three services: add_content(), retrieve(), recommend(). Remember that all three services reside in the same rest api instance, hence in the same docker container.
- docker: folder containing:
  - Dockerfile: file to create the docker image. It creates a conda virtual environment where dependencies are installed together with OpenAI CLIP github to load the model instance and run the encoding of images and texts.
  - docker-compose.yml: which build the docker image and run a container all at once by means of the "docker-compose up" command.
  - .env: file containing the PORT variable at which the service is exposed. This value can be edited to choose switch service port. Its default value is 6000.
- usage: folder containing two examples of retrieval and recommendation services usage.
- requirements.txt: list of application dependencies which is read by the Dockerfile.
- .png: images showed in the README.md
- README.md: description of exposed services: expected input and output.
- scripts: folder containing shell scripts to run the rest api.
- log: folder with a log file listing activity of the last session.

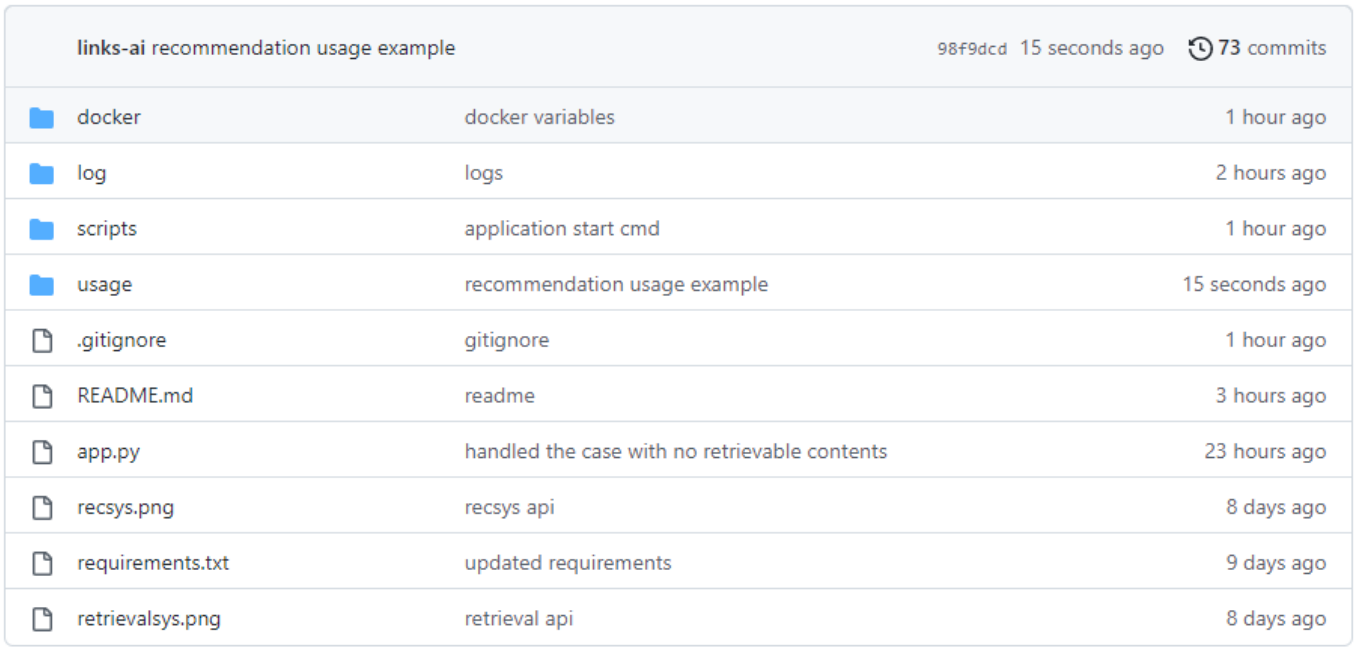


Figure 34: Project folder.

3.7.2 How to Use

- Download and install Docker Desktop available at: https://www.docker.com/products/docker-desktop/.
- Run "docker-compose up –build" to start the REST application.

4 Multimodal Recommendation Technology

In general, recommender systems can be classified in different categories based on the approach and on the different sources of information for providing users with recommendations of items. The most widespread categories of recommender systems are asset-based, collaborative filtering, knowledge-based, and hybrid (Adomavicius & Tuzhilin, 2005):

- Content-based: the user will be recommended items like the ones preferred by them in the past.
- Collaborative filtering: the user will be recommended items that people with similar tastes and preferences liked interacted with.
- Knowledge-based: these embed domain-specific knowledge that is used for matching user requirements (often expressed in a query) with items of potential interest.
- Hybrid approaches: these combine the previous methods.

All these approaches try to balance factors like accuracy, novelty, diversity, and stability in the recommendations. Each of the previously mentioned recommendation approach has its advantages and disadvantages (Table 24).

Table 24: Different approaches to Recommendation systems.

APPROACH	PROS	CONS
CONTENT-BASED	No need to cross user behaviours and profiles	Lack of cross user interactions
COLLABORATIVE FILTERING	Scalability, only based on ratings	Data sparsity and cold start problems
KNOWLEDGE-BASED	No user activity history required	Access to a structured knowledge underpinning the items
HYBRID	Inherits the advantages of the combined methods	Limited performance

Content-based recommender systems are classifier systems derived from machine learning research. These systems use supervised machine learning to induce a classifier that can discriminate between items likely to be of interest to the user and those likely to be uninteresting. The content-based approach takes as data source the user's previous interactions; therefore, it does not raise privacy issues, however as it is a supervised approach, it requires data descriptions. The collaborative filtering approach analyses the interactions of similar users for identifying candidate items. As a collaborative filtering system collects more ratings from more users, the probability increases that someone in the system will be a good match for any given new user. However, a collaborative filtering system must be initialized with a large amount of data, because a system with a small base of ratings is unlikely to be very useful. It suffers from the "cold start" problem: until there is a large number of users whose habits are known, the system cannot be useful for most users (Burke, 2000). Knowledge-based recommender systems avoid these drawbacks: for example, it does not have a cold start problem because it does not depend on a base of user ratings.

Due to the inability to track users "previous interactions with other users" content, we have decided to adopt a content-based approach that leverages user previous assets posted in the MV network. In this way, the user's

preferences do not emerge from the contents seen but from those generated by himself. This approach differs from the usual content based (as defined above) but is better suited to the MediaVerse context. Among the identified approaches in Table 24, we judged the content-based one as the most compatible with our case.

4.1 Related Work

Content-based recommender systems analyse a set of documents and descriptions (or only documents or descriptions) of items previously rated by a user, and build a model, also called profile, of user interests based on the features of the objects rated by that user (Mladenic, 1999). The profile is a structured representation of user interests adopted to recommend interesting new items. The recommendation process boils down to matching the profile built from the data and the attributes of a content object. The result of the process is the level of relevance that a new content has for the considered user. The way it creates the profile is important for the effectiveness of an information access process that assists users in finding content.

According to Lops et al. (2019), while pure collaborative filtering methods dominate the research landscape (Jannach et al., 2012), considering information about the content is highly important in many of today's practical applications of recommenders, and will be increasingly relevant in the future for mainly two reasons. First, certain aspects of a recommendation system could only be implemented in an effective way when one considers knowledge about the features of the items. These aspects, for example, include the diversification of the resulting recommendation lists, the generation of explanations, or the creation of understandable user models. Second, we continue to see new knowledge sources that provide information about the items, and which could be used in the recommendation process. These sources include both structured ones like DBpedia (Bizer, et al., 2009), semi-structured ones like Wikipedia, and unstructured ones. This latter class includes all sorts of user-generated content like reviews and tags, ultimately leading to a much richer item representations than those that were available in the past.

The use of vector space models in recommender systems has introduced a clean and solid formalism to represent content in a vector space where to perform calculations on them such as grouping similar objects or utilizing distance functions. The simplicity of such systems does not hurt the effectiveness of the model (Musto, 2010).

A seminal approach to exploit the vector space in recommender systems based on user interactions has been proposed by Agarwal et al. (2005) making use of a subspace clustering algorithm. The application of clustering approaches in recommender systems has been investigated also for reducing the sparsity of the information that is often managed when the recommendation is triggered (online). Appropriate tool to support recommender systems in increasing time efficiency are clustering algorithms, which find similarities in off-line mode (Kużelewska & Wichowski, 2015).

Recently, there is a rising attention in utilizing CLIP (Radford et al., 2021), a vector space model, for both mapping objects and retrieving objects in a recommender system implementation. Another work (Chia et al., 2022a) builds on top of the recent wave of contrastive-based methods for representational learning. CLIP-like models are still very new in this domain, another example is GradREC that leverages the space learned by FashionCLIP, a fashion-fine tuning of the original CLIP (Chia et al., 2022b).

In the context of MediaVerse, without the mechanism to keep track of the user feedback, the user profile is built using the history of the content that the user has generated. Content can be of different types such as text, image, and video, being MediaVerse a multimodal context. The use of a vector space model that operates in a

multimodal context adds a further challenge that translates into creating a recommender system operating on all modalities altogether. This is a challenge that the algorithm described below addresses.

4.2 CLIP-based recommendation algorithm

The implementation of the recommender system developed for this project exploits the CLIP-based retrieval system of Section 3. While in the retrieval system the seed for the retrieval is a query (image/text) given by the user, in the recommendation system there is no user input. Furthermore, there is not a history of user interactions and there is not a user's feedback for the recommendations to adjust them. Therefore, to create the seeds for the retrieval, the algorithm makes use of the previous posts that the same user has made.

4.2.1 System Architecture

This section describes the system architecture of the recommendation service. The recommendation task does not require any input from the user, but a request for recommendation must be sent to activate the recommendation system. This is composed of two main classes: the container and the Faiss indexer. Each class was instantiated twice, for images and texts. The container structure is the same as in Figure 20, described in section 3.3. The role of the container is to simulate the presence of a database in the REST API implementation. Meanwhile the Faiss indexer is the same component previously described in 3.3.1. In Figure 35, the system architecture is illustrated. Like the retrieval system, the user can access the system using two actions:

- **Post content**: in this way the user can add a content (text or image) to the system. It will be encoded to a 512 embedding vector and stored into the Faiss database. When adding an embedding to the Faiss database, the relative Faiss index associated with that content is returned and stored in the container, along with the relative content id and the username of the user who posted.
- **Recommendation request**: when the user asks for recommended contents, the system performs the following steps:
    - Retrieves the previously posted contents, given the username of who requested the service.
    - Based on the user's previous contents, the recommendation system builds a series of seeds, with an algorithm described in Section 4.2. 2..
    - The recommendation seeds are used as inputs to the retrieval system, that searches the top k most similar contents among the MediaVerse content (excluding those posts that belong to the user itself).

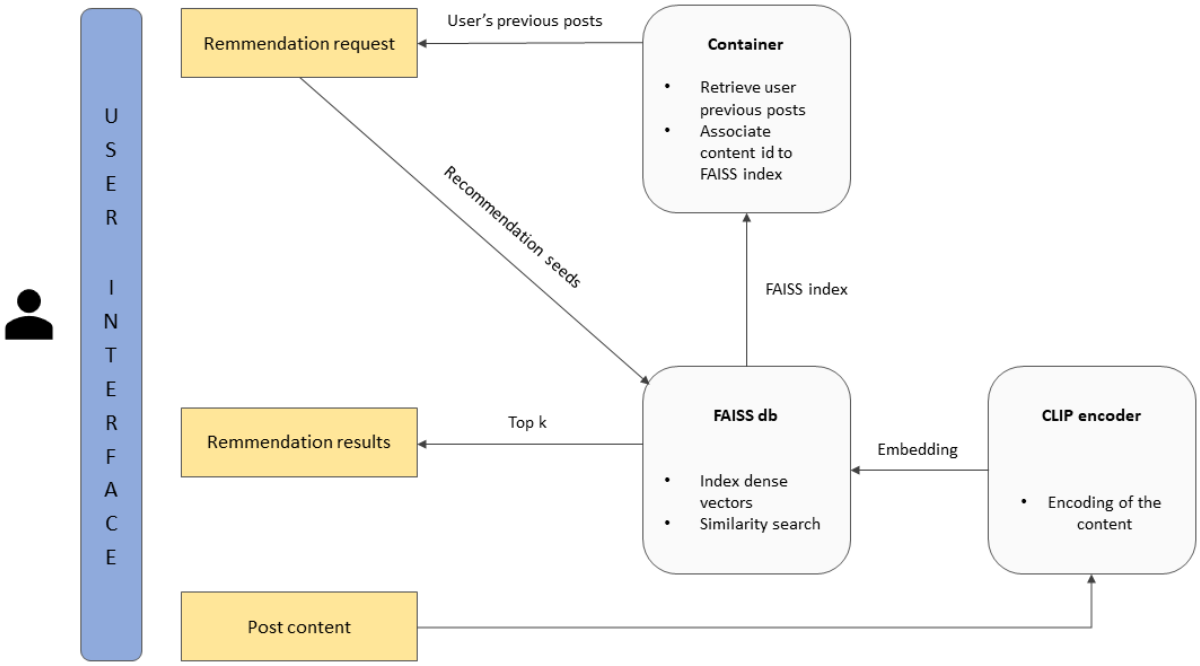


Figure 35: System architecture.

4.2.2 Algorithm

This Section describes the algorithm that allows the system to retrieve the user's previous posts and generate seeds for the retrieval system. As mentioned earlier, when a user asks for recommended content, the system retrieves the Faiss indexes of the user's previous posts from the container, by accessing the value of the dictionary associated to the username key. From the retrieved Faiss indexes, through the function `index.reconstruct(idx)`, the embeddings of the posts are reconstructed.

Since the number of previously posted images or texts is not known a priori, one challenge of this task was building a way of having a reduced number of seeds to use as input to the retrieval system. One possible solution is a clustering of the embeddings, to "group" all the user's previous posts into a number of interest areas and select the most relevant ones.

The clustering technique chosen for this task was HDBSCAN. HDBSCAN stands for Hierarchical Density-Based Spatial Clustering of Applications with Noise. This technique uses density of neighbouring points to construct clusters, allowing clusters of any shape to be identified. After the clustering of the user's previous posts, three cases are distinguished:

1. Number of clusters = 0:
    a. Retrieval of k similar contents for each embedding of the user's previous posts, so each content is a seed to the retrieval system.
    b. Selection of k random elements, with a probability distribution function built on the similarity scores, with a softmax activation function.
2. Number of clusters = 1:
    a. Selection of the cluster medoid and use of it as seed to the retrieval system. Return of 80% k.
    b. Selection of a random outlier and use of it as seed to the retrieval system. Return of 20% k.

3. Number of clusters >= 2:
    a. Selection of the medoids of the two main clusters and use them as seeds to the retrieval system. Return of 40% k each.
    b. Selection of a random outlier and use it as seed to the retrieval system. Return of 20% k.

The values of fraction of k returned from the outlier seed is chosen arbitrarily, according to a trade-off discussed in Section 4.4.2. In the cases 2) and 3) the point chosen to represent the cluster is the medoid. The medoid is a point that is part of the cluster and that has minimum distance from all other points. The mean or the centroid are points that in the HDBSCAN clustering technique could result in being far from the actual cluster, because for HDBSCAN the clusters might not be round, since it is a density based and not special based technique. The resulting clusters might have a convex shape. Therefore, the centroid of the cluster might be in the area of another cluster, thus misleading the recommendation and creating a non-representative seed.

4.3 Experimental Setup

Here, we describe the setup of the experiments that were ran to analyse the recommender system. We average the results of 100 iterations, performed with a total of 5000 samples of the validation 2014 split of the MSCOCO dataset, which correspond to 1114 image/caption couples. The samples are divided among two users:

- The first representing the user who makes the recommendation request, called the *posting user.*
- The second representing the rest of MediaVerse content items, i.e., the possible content that can be recommended.

Two types of analysis were made to evaluate the performance of the recommendation system: the qualitative analysis and the quantitative analysis. In the qualitative analysis we will show the input, the output and how the seeds for the recommendation are generated, allowing the reader to evaluate with visual inspection the recommended content with respect to the seeds. In both types of analysis two cases are distinguished:

- Case number of clusters = 0, evaluates the performance of the random selection.
- Case number of clusters ≥ 1, evaluates the performance of HDBSCAN clustering for generating the seeds.

In the quantitative analysis the main parameter that will be used for the evaluation will be the one of the diversity indexes. The diversity is calculated with the Intra-List Distance (ILD), defined as the average pairwise distance among items (Antikacioglu et al., 2018).

The distance *dist(v_k, v_j)* between items is measured using the Euclidean distance between the items' embeddings. Given a list *L* of recommendations, defined by item lists of length c_u for user *u*, the intra-list distance is defined as follows:

$$ILD = \frac{1}{|L|} = \sum_{u_i \in L} \frac{1}{c_i(c_i - 1)} \sum_{(v_k, v_j) \in N(u_i)} dist(v_k, v_j)$$

A useful parameter introduced in this section to evaluate the diversity is the parameter *d*, which is the fraction of k recommended from the seed extracted from the outliers. This parameter can only be used in the second case (number of clusters ≥ 1).

Usually, to evaluate a recommendation system, useful parameters to consider, other than the diversity, are serendipity and novelty. The novelty is a measure of how many new contents are present among the recommended ones, while the serendipity is the measure of how much a new content is appreciated by the user.

However, evaluating these parameters would require the presence of a user experience for feedbacks and for recording of the contents that have already been seen. The ILD was chosen as diversity metric because it allows us to evaluate the diversity between the recommended items, without involving the user's experience.

4.4 Results and Discussion

Here, we present the qualitative and quantitative analysis and discuss the results. Furthermore, we draw conclusions from the results, highlighting the limitations of the system and proposing future improvements.

4.4.1 Qualitative Analysis

Here, we provide examples of the functioning of the recommendation system. For this purpose, we use the MSCOCO validation 2014 split, with 5000 samples. From 5000 we select 1114 image/caption couples, and we cluster the images with HDBSCAN, obtaining 7 clusters. Figure 36 presents the clustering of the images with a 2D projection, obtained with TSNE python library. We assume that the caption associated to an image belongs to the same cluster as the image itself.

Figure 36: 2D representation of 1114 samples of the MSCOCO validation 2014 split clustered with HDBSCAN.

We divide the 1114 samples between two users: one representing the posting user and the other representing the whole MediaVerse contents, as described in Section 4.3. For the first case, where we want to simulate the event in which HDBSCAN is not able to find any cluster, we randomly extract few samples, while to simulate the second case, we extract a larger number of samples that are known to belong to one of the previously formed clusters, as we will describe in the next examples.

Number of clusters = 0

The case in which HDBSCAN is not able to find any cluster, it might happen when the user's posts are too few to allow the creation of clusters. Therefore, for this setup, we uploaded as user's previous posts 16 samples for each type, images, and texts, of the same couple image-caption and we output the top k recommended content, with k = 10. Figure 37 shows the texts that represent the user's previous posts, while in Figure 38 shows a representative example of the three main topics from which these images were extracted: airplanes, motorcycles, and streets.

```
Commercial liner at a airport parked on a runway.
A line of jetliners parked next to each other at an airport.
Four airplanes are flying over a grassy area.
Airplanes at the airport getting ready to take off.
A large jet takes off from an airport runway.
Clear blue sky's with a plane flying by.
There is a jet plane that is painted multi color
White jet plane flying in the sky with engines in the wings.
A large airplane flying overhead in the sky
A stream of smoke from a jet in the sky.
A 787 Jet Airplane sitting on a runway at an airport.
A commercial airplane flies through the sky to its destination.
Traffic light at night, appearing very confusing.
A man riding a motorcycle down a busy city street.
A black  motorcycle parked outside of a house.
A black motorcycle parked in the dirt by the beach
```

Figure 37: Captions posted by the user.

In this case, HDBSCAN is not able to find any cluster because of the limited number of input contents, therefore each image and each text is used as seed to the retrieval system. Figure 39 presents the output of the random selection, according to a distribution based on the similarities. The IDs of text2image output corresponds to the images, in Figure 40. The recommended contents respect the three main themes of the input images. The same holds for the image2image output (Figure 41). The image2image recommended asset contains the three main subjects of the user's previous post: motorcycles, planes, and a street image.

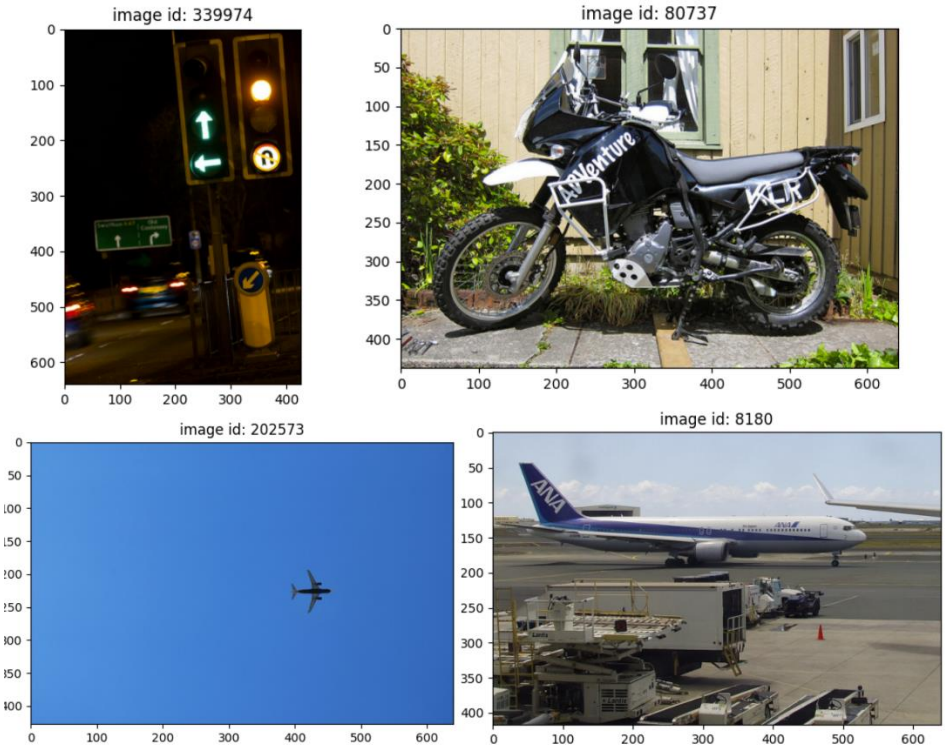


Figure 38: Example of four images posted by the user.

```
text2text    :    content:      A Silver Plane on a snow covered runway
                                A street view with a plane flying ahead
                                A white and blue jet is at the airport
                                A street filled with lots of traffic next to tall buildings
                                A silver color jet flying through the clear sky
                                Airplanes on an airstrip beneath a hazy sky
                                A motorcyclist pauses to enjoy a view of mountains
                                A blurry motion picture of a person doing a wheelie on a motorcycle
                                People stand around an antique motorcycle in a grassy area
                                The woman rides her bike down a small alleyway
                     scores:    0.72441566, 0.80601966, 0.69219154, 0.65885144, 0.77607673, 0.80865926,
                                0.67255956, 0.86733836, 0.80267763, 0.796427

text2image   :    content:      440124, 546642, 459912, 368541, 468508, 566038, 488327, 507362, 392650, 456865
                     scores:    0.28296635, 0.280 04947, 0.277429, 0.27343008, 0.28671747, 0.29526785, 0.23261306,
                                0.2857124, 0.25988728, 0.28784853

image2text   :    content:      A man pointing at a police motorcycle and officer
                                An image of a motorcycle parked next to a fence
                                An airplane tarmac with planes and trucks at sunset
                                A person in a helmet standing by their motorcycle
                                Ladies riding down the street on motorcycles as others look on
                                some planes on the run way of an air port
                                A man in posing with a motorcycle in front of a bay
                                A mountain biker pumps his fist in celebration
                                A parked motorcycle on display next to the skull of a demon
                                A multi-colored airplane makes its way through the blue sky
                     scores:    0.31676367, 0.29186732, 0.2567752, 0.2752496 , 0.29521334, 0.3008315,
                                0.29367656, 0.30401307, 0.27037027, 0.32076818

image2image  :    content:      412184, 262394, 422326, 96549, 338325, 61259, 56070, 383065, 422326
                     scores:    0.6951229, 0.7433229, 0.7384074, 0.74724823, 0.7280412, 0.7083852, 0.7403492,
                                0. 74760723, 0.7173965, 0.7384074
```

Figure 39: Output of the recommendation system in the "zero clusters" case.

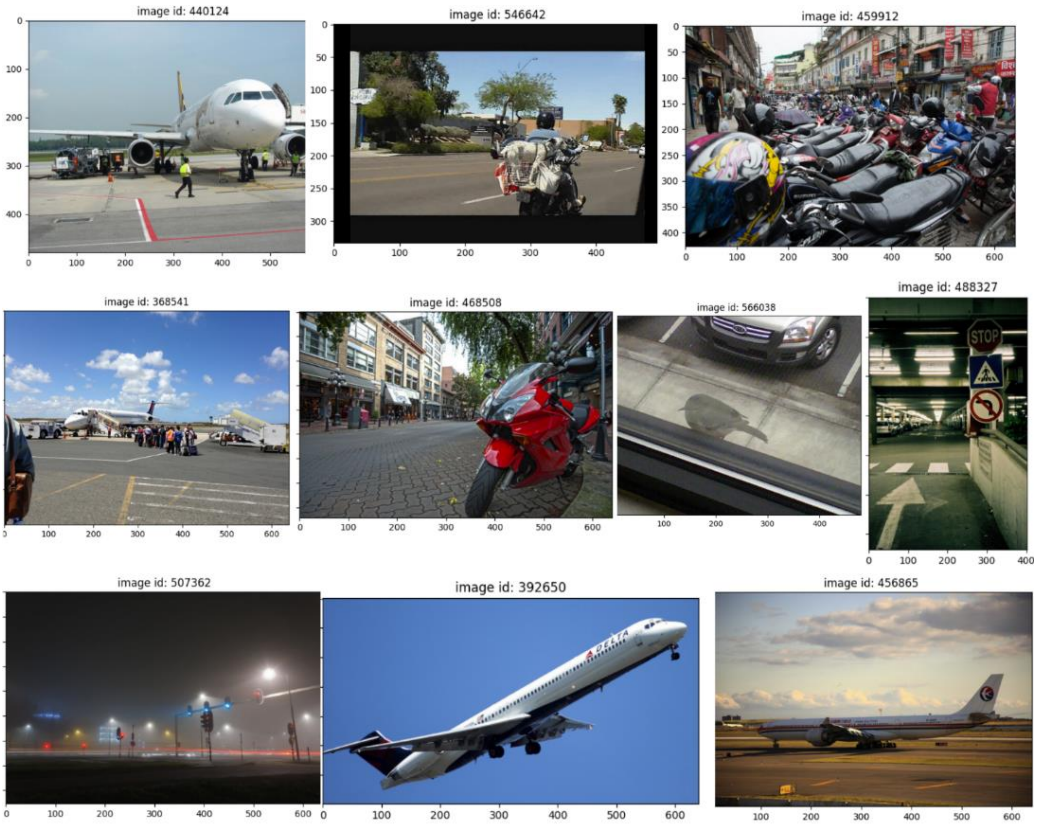


Figure 40: text2image recommendation.

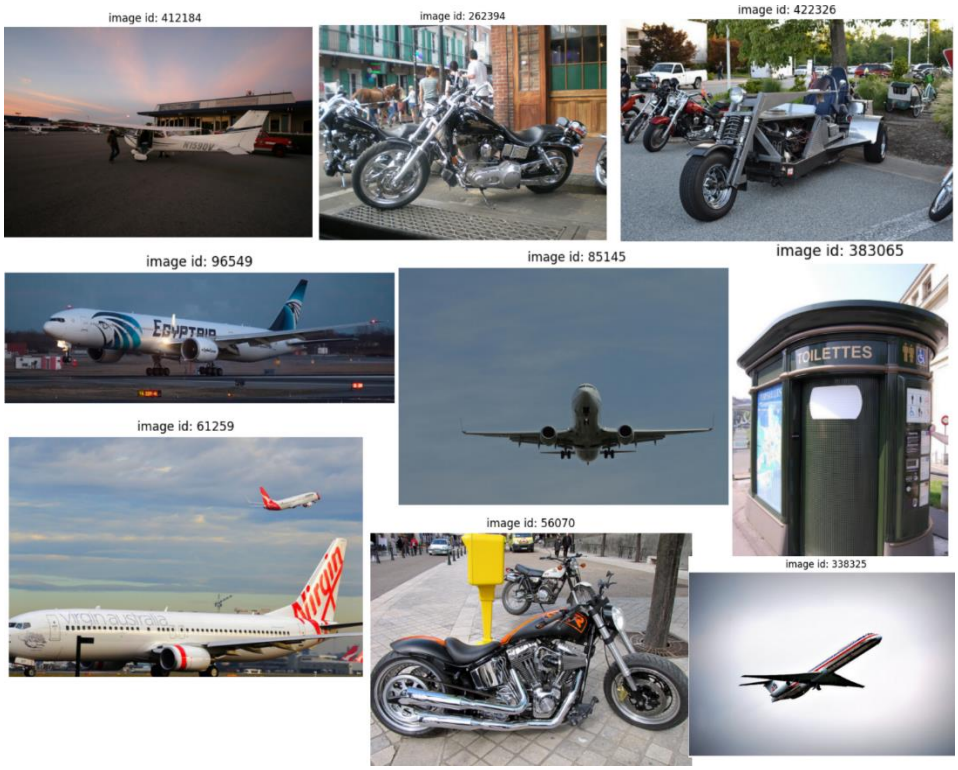


Figure 41: image2image recommendations.

Number of clusters > 0

The case in which the posted content of the requesting user is grouped by HDBSCAN into more than one cluster represents the case in which the user has enough previous posts to allow HDBSCAN to create clusters. In this case we return 80% of k from the main clusters and the remaining 20% from a random outlier, allowing the output to vary from the user's main interests.

We analyse an example in which the user posted a total of 240 images/captions couples, of which 129 are taken from a cluster representing bathrooms, 10 from the cluster representing wild animals, 30 representing planes and 71 are taken from random outliers. These samples are then clustered by the recommendation system (Figure 42). The 2D representation of the clustering was performed with TSCAN library. For the texts, 3 clusters were found, while 4 clusters are found for the images. Therefore, even though the clustering was performed with the same parameters, the type of the content influences the number of clusters found.

In this case, the seeds given as input to the retrieval system are selected from the medoids the two main clusters (one if there is a single cluster) and one from a random outlier.

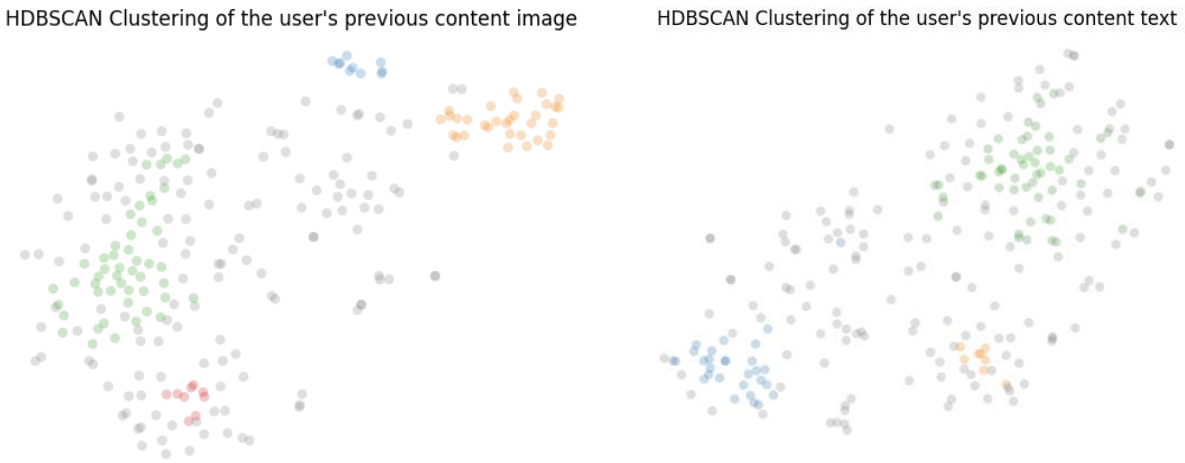


Figure 42: Clustering of the user's previous posts.

Figure 43 shows the selected seeds. As expected, the seeds from the two main clusters represent bathrooms and planes for both images and texts. Meanwhile the randomly selected outlier seeds represent a living room and a street, in texts and images respectively.

```
Chosen seed for the outliers:  ['A living room with a large tv and pictures all around.']
Chosen seed for the cluster 1:  ["A bathroom with a toilette with it's seat down."]
Chosen seed for the cluster 2:  ['A commercial airplane flies through the sky to its destination. ']
```

```
Chosen seed for the outliers:  ['75560']
Chosen seed for the cluster 1:  ['242363']
Chosen seed for the cluster 2:  ['57429']
```

Figure 43: Seeds extracted from the clustering.

Figure 44 shows the output of the recommender system in this case. For the text2text recommendations, as well as for the text2image, the three main subjects of the output are planes, bathrooms and living room.

```
text2text    :    content:     A bathroom with a sink and a toilet
                               This is an aerial view of the nearly-finished bathroom in a house still under construction
                               A small dirty bathroom with a simple sink
                               A person took a picture of a bathroom from the doorway
                               An airplane is flying high in a cloudy sky
                               A plane that is fliying in the air
                               A photograph taken from an airplane window while in flight
                               A airplane banking to the right as it ascends into the sky
                               A contemporary living room features leather forniture, oak storage and garden view
                               A room contains baskets and has a fierplace

                    scores:    0.99642646, 0.9638996, 0.9579913, 0.9569446, 0.94775045, 0.93059397, 0.9099555,
                               0.904182, 0.8692718, 0.7998567

text2image   :    content:     61422, 358606, 230610, 253441, 564003, 153445, 467475, 175942, 41974, 338325
                    scores:    0.33534095, 0.3318021, 0.32373518, 0.31776345, 0.3159011, 0.30069673, 0.30053076,
                               0.29135072, 0.2907191, 0.2888479

image2text   :    content:     A picture of a bathroom with a toilet, rug, and shelving on the door
                               A tiled bathroom with white towels hanging up
                               A small bathroom with a small brown toilet next to a white sink
                               A small white toilet sitting next to a metal trash can
                               Big commercial airplane getting ready to take off
                               A Kenyan Airwas airplane sits on the runway
                               A stop light has a blue sign with a P
                               An airplane just landed on the runway
                               A plane photographed in the blue, cloudless sky
                               A group of traffic lights sitting above an intersection
                    scores:    0.3321036, 0.33713493, 0.33694613,0.33323234, 0.30174434, 0.29812455, 0.29412284,
                               0.2919977, 0.28808668, 0.2708894

image2image  :    content:     495984, 85145, 368349, 560591, 443652, 36492, 440124, 419074, 195267, 346904
                    scores:    0.94010216, 0.93671155,  0.9310354, 0.924188, 0.9210861, 0.831623, 0.8263519,
                               0.8250364,0.7181187, 0.71581316
```

Figure 44: output of the recommender system with clustering.

The same holds for the image2text and image2image recommendations, where the three main subjects present in the recommended items respect the subjects of the image seeds. Figure 46 shows the output of the image2image recommendation: planes, bathrooms, and streets.

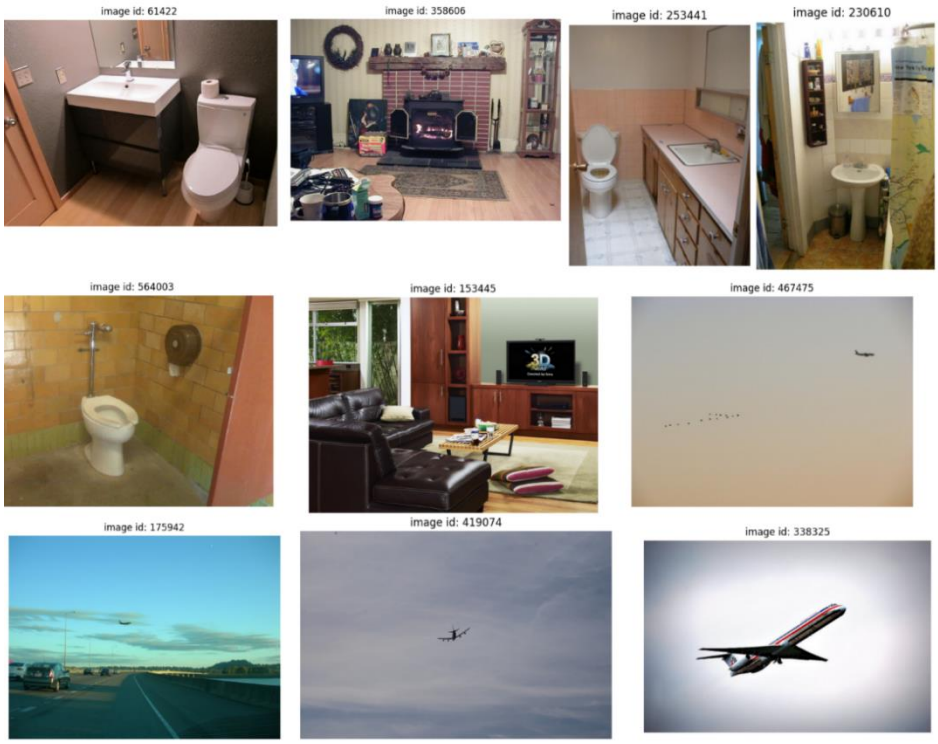


Figure 45: text2image recommendations.

Figure 46: image2image recommendations.

4.4.2 Quantitative Analysis

Here, we report the variation of the ILD (Intra List Distance, described in Section 4.3) according to two parameters: the number *k* of recommended contents, and the fraction *d* of content that comes from the outlier seed. Moreover, we will analyse how the recommendation system execution time scales with the number *n* of posted contents by the posting user. Summarizing, we have three parameters:

- *k*: the number of recommended contents
- *d*: the fraction of contents that are generated from the outlier cluster
- *n*: the total number of contents posted

ILD vs k

To analyse the diversity with respect to k we chose several posted content *n* first equal to 20, to simulate the case in which previously posted contents form no cluster due to its scarcity, and then *n* = 200 to simulate the actual clustering scenario (Figure 47).

In the first case, as expected, the diversity proportionally increases with the increase of *k*. In the case in which the system is not able to generate any cluster from the previous posts of the user, each of the previous post is a seed. For example, if k = 10, it means that the output includes recommended contents from at most 10 seeds, but if k = 100 it means that the output can include at most items recommended from 100 different seeds. Therefore, as k increases, the number of different seeds employed is likely to increase as well. This results in higher diversity among the recommended contents.

In the second case, the number of recommended contents *k* influences the diversity only for values below *k* = 20. This happens because if we have less recommended content from three different areas of knowledge (two

seeds for the two main clusters and one from outliers) the pairwise distance between the recommended contents will be higher. If we have more recommendations from the same area of knowledge the average pairwise, distance between the recommended items will be smaller.

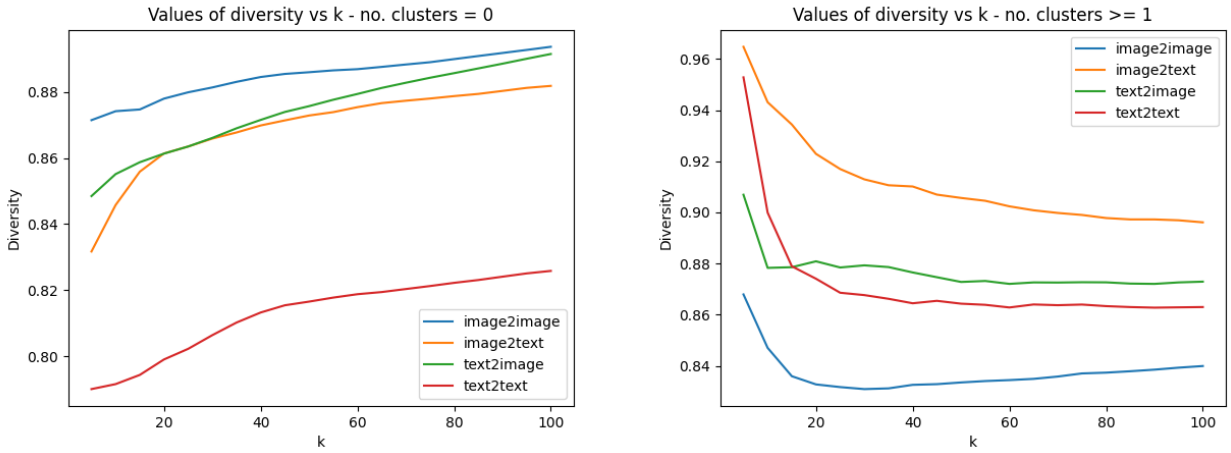


Figure 47: Variation of ILD according to k in the case with no clusters and the case with more than one cluster.

ILD vs d

We chose the number of posted contents *n* equal to 200 to simulate the clustering scenario, and *k* = 100. The range of *d* varies from 5% to 95% (Figure 48). In this case, the ILD increases with the percentage of content returned from the outliers. It is worth mentioning that both in this and in the previous case the output of a cross-modal recommendation has a higher ILD with respect to the output of a uni-modal recommendation. This means that, for example, the recommended images retrieved from the previously posted texts have higher diversity than the recommended images retrieved from previously posted images. Therefore, the cross-modal recommendation naturally introduces diversity in the output. The value of *d* can be selected with a trade-off between diversity and relevance of the chosen outlier with respect to user's main interests. Therefore, as stated in Section 4.2.2, the chosen fraction of *k* returned from the outlier is 0.2.

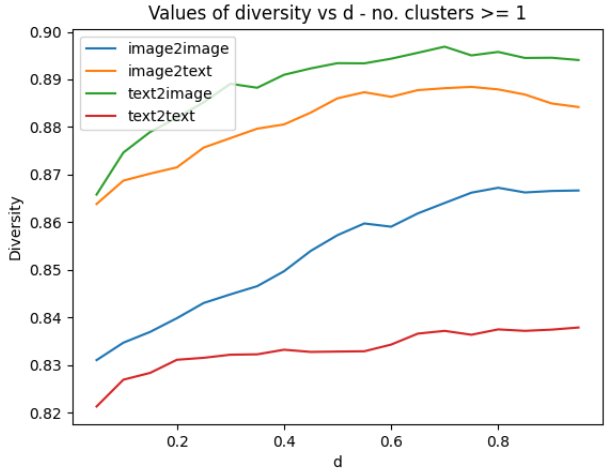


Figure 48: Variation of ILD according to d.

4.4.3 Conclusions

From the previous analysis, it becomes evident that the parameters of the system have a limited influence on the range of values of the diversity that remains between 0.80 and 0.99. The reason is in the way the system was built independently from the number of posted contents and the number of recommended contents, the system selects only three main seeds to represent three areas of interest, which is the main limitation of the system.

Limitations

The main limitation arises from the fact that, regardless of the number of previously posted contents, the system selects only three seeds. The two seeds from the two main clusters may not provide a complete representation of all the interests of the user. It is also important to mention that the algorithm suffers with the so-called "cold start" problem: if the user has not yet posted, but it requires recommended content, the system will not be able to generate seeds and therefore recommend content.

Future work

Future improvements of the recommendation system can include an interaction of new users with the MediaVerse platform, in which the platform itself suggests some images to the users and asks them to select images of their interest, to help the system get over the "cold start" phase.

Another improvement of the system can include increasing the number of seeds with the number of generated clusters from the previously posted content, in order to increase the diversity of the recommended items.

4.5 Design and Deployment

The recommendation service is exposed by the recommend() function which resides on the same REST API of the retrieval system (Figure 49). Hence, the application has only one docker container exposing 3 functions: add_content(), retrieve, recommend(). In the recommendation context it is assumed that the user who add contents in the MV node is different from the one who receives the recommendation. Different users are identified by their username. The user can access the system by means of two functions:

- **add_content()**: in this way the user can add content (text or image) to the system. It will be encoded as a 512 embedding vector and stored into the Faiss index (same as retrieval system in 3.7).
    - Input:
        - username: unique identifier of the user who post the content.
        - text or image binary data (refer to usage.py example).
        - id: Mediaverse ID of the content to load.
        - type: "text" or "image", string describing the data type of the content to be loaded.
    - Output:
        - a success message in the field "msg"
        - the elapsed time for the operation in the field "time"
- **recommend():** the user does not enter any query. Starting from a seed, the System suggests to the user new contents based on user post history and a certain degree of novelty. The top K contents are ranked according to the cosine similarity with respect to the generated seed.
    - Input**:**
        - username: unique identifier of the user who search for a content (its contents are excluded from the recommendation process).

- k: number of similar assets to recommend.

- Output (divided into 2 sections):
  - "text" (if the user seed is build starting from text contents):
    - (text2text) - recommended texts in the field "text":
      - "assets" is an ordered list containing the ids (string) of the retrieved texts. The list is ordered based on decreasing values of similarity scores (i.e., the first asset is the best one retrieved for that query).
      - "scores" is an ordered list of similarity scores for the retrieved texts.
    - (text2image) - recommended images in the field "image":
      - "assets" is an ordered list containing the ids (string) of the retrieved images. The list is ordered based on decreasing values of similarity scores (i.e., the first asset is the best one retrieved for that query).
      - "scores" is an ordered list of similarity scores for the retrieved images.
  - "image" (if the user seed is built from images):
    - (image2text) - recommended texts in the field "text":
      - "assets" is an ordered list containing the ids (string) of the retrieved texts. The list is ordered based on decreasing values of similarity scores (i.e., the first asset is the best one retrieved for that query).
      - "scores" is an ordered list of similarity scores for the retrieved texts.
    - (image2image) - recommended images in the field "image":
      - "assets" is an ordered list containing the ids (string) of the retrieved images. The list is ordered based on decreasing values of similarity scores (i.e., the first asset is the best one retrieved for that query).
      - "scores" is an ordered list of similarity scores for the retrieved images.

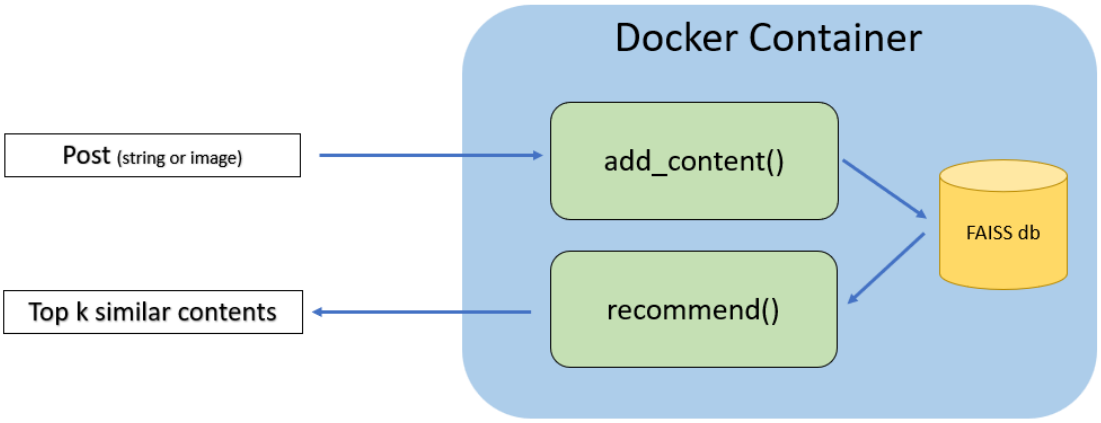


Figure 49: Rest API - Recommendation.

4.5.1 Project Folder

Refer to 3.7.1.

4.5.2 How to Use

Refer to 3.7.2.

5 Services for the Automatic Adaptation of MediaVerse Assets

5.1 Introduction

In recent years, the advances in the computing capacity of modern compact devices have been truly remarkable, but at the same time, newly created media experiences require increasingly more powerful devices. Also, the introduction of 5G networks opens the possibilities for real-time media experiences. But still 5G is not fully extended and accepted by all kinds of devices in the real world. Therefore, modern applications are intended to be compatible with 5G networks, but they are not optimized to be transmitted over these networks.

The above give rise to the intense engineering work that is necessary to adapt the original media files to others that are more portable, lightweight, or consumable by any type of user. Therefore, research into new methods to compress files, or the search for standardized media types is still highly relevant.

ATOS explores the most modern asset adaptation methods available by testing on real-world media assets, reaching conclusions on the best available optimizations for each type of file.

5.2 Asset Adaptation Pipeline

T3.3 takes care of developing resources for the automatic adaptation of asset. Figure 50 represents the first approach on the adaptation of asset that was made available in the platform.

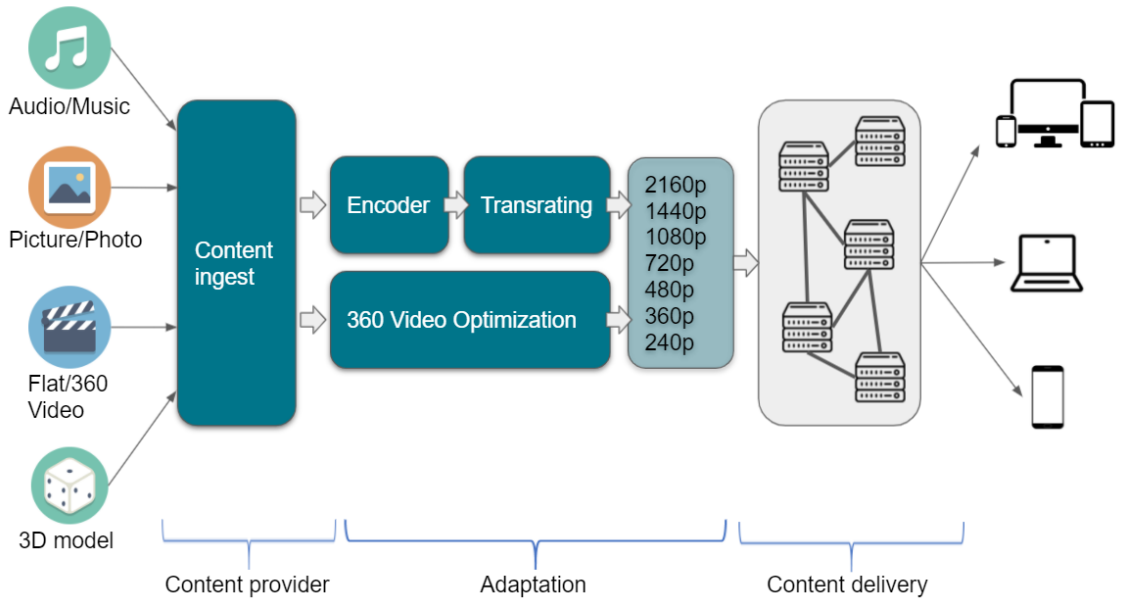


Figure 50: Base asset delivery workflow.

During the development and testing phase, it became clear that this approach is incomplete, since it represents all data types passing through the same typical video processing components. Each MediaType file is completely different, so a separate workflow is needed to transform each of them. In addition, as the first phase of the user interface for the platform was developed, the need to offer different transformations for different user use cases became noticeable. For example, a video shown to a user to preview, or appear in a list of assets quickly, should

not have the same quality as the view for the owner/editor of the asset. The platform should provide an approach for every asset representation on the UI. For this reason, the platform needed to evolve to a different one in which all types of media files are optimized.

To summarize, the following are the media data types that the consortium has agreed to accept on the platform during this development phase:

- Text
- Pictures
- Audio
- Video
- 360 Video
- 3D models

The status of the transformations available for each type of file will therefore be separated and explained in a dedicated section. Moreover, through the first tests of the interface, we have detected the following representation needs for each type of asset:

- The original asset.
- An optimized version for the owner / purchaser to visualize the asset.
- A previsualization version for non-owners and viewers from external nodes.

As MediaVerse is a decentralized platform, the publication of the transformation results led the consortium to a discussion about the availability of the assets for users of other nodes, or non-owners of the asset. The original assets should not be shown to users unless they own them or have rights to use them.

The following table (Table 25) represents the approach to the representations and the publication method of an asset that should be available for the different use cases.

Table 25: Required transformations for each Media Type.

	Search View	Asset view			Original	
Media Type	ALL	OWNER	PURCHASER	OTHER USER	OWNER	PURCHASER
Text	Description and generic Thumbnail	Simple text editor	Text viewer	description	Download/ Delete	Download
Pictures	Compressed version with watermark	Original	Original	Compressed version with watermark	Download	Download
Audio	Description and generic Thumbnail	Original	Original	Compressed version with audio watermark	Download	Download
Video	Thumbnail	Adaptive Streaming version	Adaptive Streaming version	GIF summary	Download	Download
360 Video	Thumbnail	Adaptive Streaming version	Adaptive Streaming version	GIF summary	Download	Download
3D Models	Thumbnail	Original	Original	GIF summary	Download	Download

The implementation of IPFS and the federated search function enabled platform decentralization. Users should be able to list and purchase assets from other nodes. This new functionality introduced the need for a kind of a CDN to serve the asset to the users. As IPFS technology can store and serve assets in a decentralized way. ATOS proposed to use this functionality to offer some of the single file previews for external node users. If an asset is purchased from an external node, it is duplicated in the purchaser node, with MediaVerse nodes acting as a CDN.

As in previous deliverables, in the following sections, we describe the necessary processes to ensure the correct rendering of assets by the end user. The core of all these transformations is the transcoder service, of which the implementation will also be described in detail.

5.3 Transformation Workflow

The MediaVerse platform is decentralized and based on microservices. Therefore, the communication between components of the platform will be done through the APIs exposed by each of them. The components related to the transformation workflow are the UI, the transcoder, the publisher, the IPFS node and the DAM.

5.3.1 Architecture

The main purpose of this section is to explain the integration of the transcoding service with the rest of the platform to the internal components of the service.

Integration

The transformations required to display in the interface have been approached from three use case perspectives: An asset creator, a buyer, and an external user, either from the node or from an external node, are searching for assets. Figure 51 illustrates the ingest and display process for different types of users, isolating only the interaction between the DAM and the transcoding service. This process is also described in detail below.

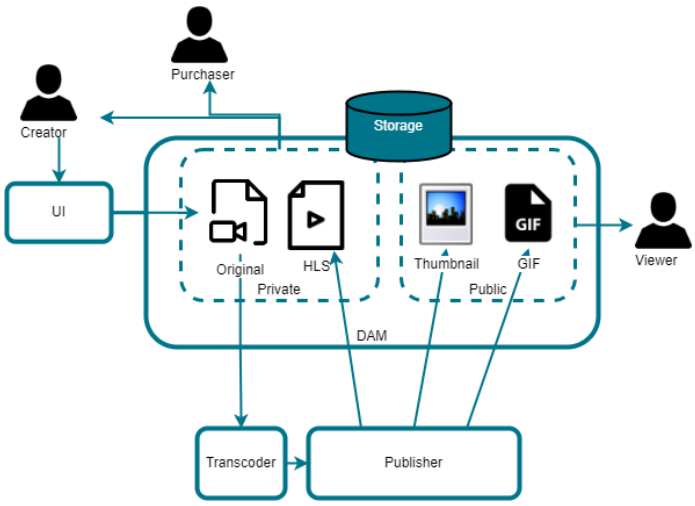


Figure 51: Transcoder interaction with other MediaVerse services.

1. The asset creator uploads any media asset using the interface.
2. Both from the interface and from the DAM the mime type of the asset is filtered, and the DAM makes the necessary calls to the transcoding service.
    a. The DAM uses the “Direct transformations” API endpoints to generate the pre-visualization transformations for the asset: Thumbnail and GIF

b. The interface shows the user that the asset has been uploaded, and direct transformations, GIF or Thumbnail. The asset is now ready to be searched, or its metadata edited in the "My assets" section.
c. The DAM calls the "Transcode and Publish endpoints" to start the adaptive transformation of the asset. This process may take a while depending on the asset uploaded.
d. Once the adaptive streaming files are generated, the transcoder informs the DAM and sends the result.

3. The owner is now able to play the video file quickly, regardless of the size, length, or device on which it is displayed.
4. If another user purchases the video, the purchaser will have the same viewing options, but won't be able to modify or delete the asset.

Internal Architecture

As shown in Figure 52, the service architecture is based on docker, making integration and deployment tasks with the rest of the components easy. The reasons for the choice of each of the components that made up the transcoder and the publisher are as follows.

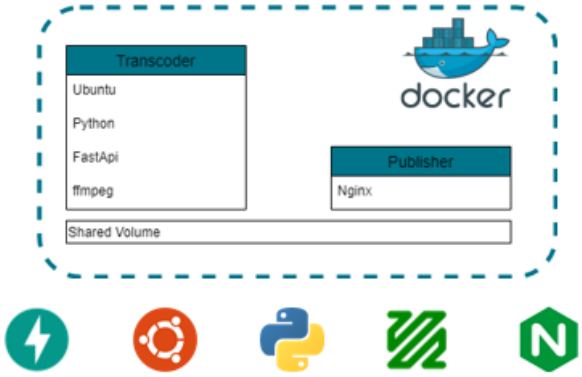


Figure 52: Transcoder service internal Architecture.

Ubuntu docker Image: "Ubuntu is a Debian-based Linux operating system that runs from the desktop to the cloud, to all your internet connected things" . It is the world's most popular operating system across public clouds and OpenStack clouds. It is the number one platform for containers, from Docker to Kubernetes to LXD[29] .It was decided to use Ubuntu as the container for the operating system, and no other more lightweight ones like Alpine due to possible compatibility issues/updated repositories of the second one. Using ubuntu made integration with the more modern and experimental features of ffmpeg easy, as both solutions are kept up to date and are fully compatible.

Python: Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built-in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. [30] Since ffmpeg is a command line tool, a versatile programming language was needed that had many options for integrating an API with operating system functions.

[29] Docker Hub. (2022). Retrieved 26 May 2022, from https://hub.docker.com/_/ubuntu

[30] What is Python? Executive Summary. (2022). Retrieved 26 May 2022, from https://www.python.org/doc/essays/blurb/

Ffmpeg: "FFmpeg is the leading multimedia framework, able to decode, encode, transcode, mux, demux, stream, filter and play pretty much anything that humans and machines have created". It supports the most obscure ancient formats up to the cutting edge. No matter if they were designed by some standards committee, the community, or a corporation[31]. When looking at the options available for transcoding videos in a backend service, there are not many more than ffmpeg. ffmpeg is the basis of almost any modern transcoder, and it offers a large number of available configurations.

FastAPI: is a modern, fast (high-performance), web framework for building APIs with Python 3.6+ based on standard Python type hints. The key features are: Very high performance, on par with NodeJS and Go (thanks to Starlette and Pydantic) . One of the fastest Python frameworks available. Increase the speed to develop features by about 200% to 300%. * Reduce about 40% of human (developer) induced errors. Great editor support. Completion everywhere. Less time debugging. Designed to be easy to use and learn. Less time reading docs. Minimize code duplication. Multiple features from each parameter declaration. Fewer bugs. Get production-ready code. With automatic interactive documentation. Based on (and fully compatible with) the open standards for APIs: OpenAPI (previously known as Swagger) and JSON Schema.[32] Choosing Fast-Api as the framework for the transcoder API was a success. Fast-Api is the most modern technology for building APIs available in Python, resulting in clear documentation for the rest of the consortium and amazing processing speed and scalability.

Nginx: NGINX is open-source software for web serving, reverse proxying, caching, load balancing, media streaming, and more. It started out as a web server designed for maximum performance and stability. In addition to its HTTP server capabilities, NGINX can also function as a proxy server for email (IMAP, POP3, and SMTP) and a reverse proxy and load balancer for HTTP, TCP, and UDP servers[33]. Using the official nginx docker image as a web server for the adaptive streaming has allowed a simple installation and a really good performance, being designed for streaming, and having specific configuration options for it.

The previous image shows the Swagger interface generated by Fast API and exposed to be used by any developer trying to understand its use. The Interface is fully functional allowing the users to directly try the endpoints and get curl examples of the calls. The endpoints have been divided into several sections depending on the nature of the transformation: Transcode and publish, to generate the adaptive streaming for a video. Direct transcoding, for the generation of previews and audio extraction and 3D Direct transcoding for the generation of a thumbnail from a 3D model.

[31] About FFmpeg. (2022). Retrieved 26 May 2022, from https://ffmpeg.org/about.html
[32] FastAPI. (2022). Retrieved 26 May 2022, from https://fastapi.tiangolo.com/
[33] What is NGINX? - NGINX. (2022). Retrieved 26 May 2022, from https://www.nginx.com/resources/glossary/nginx/

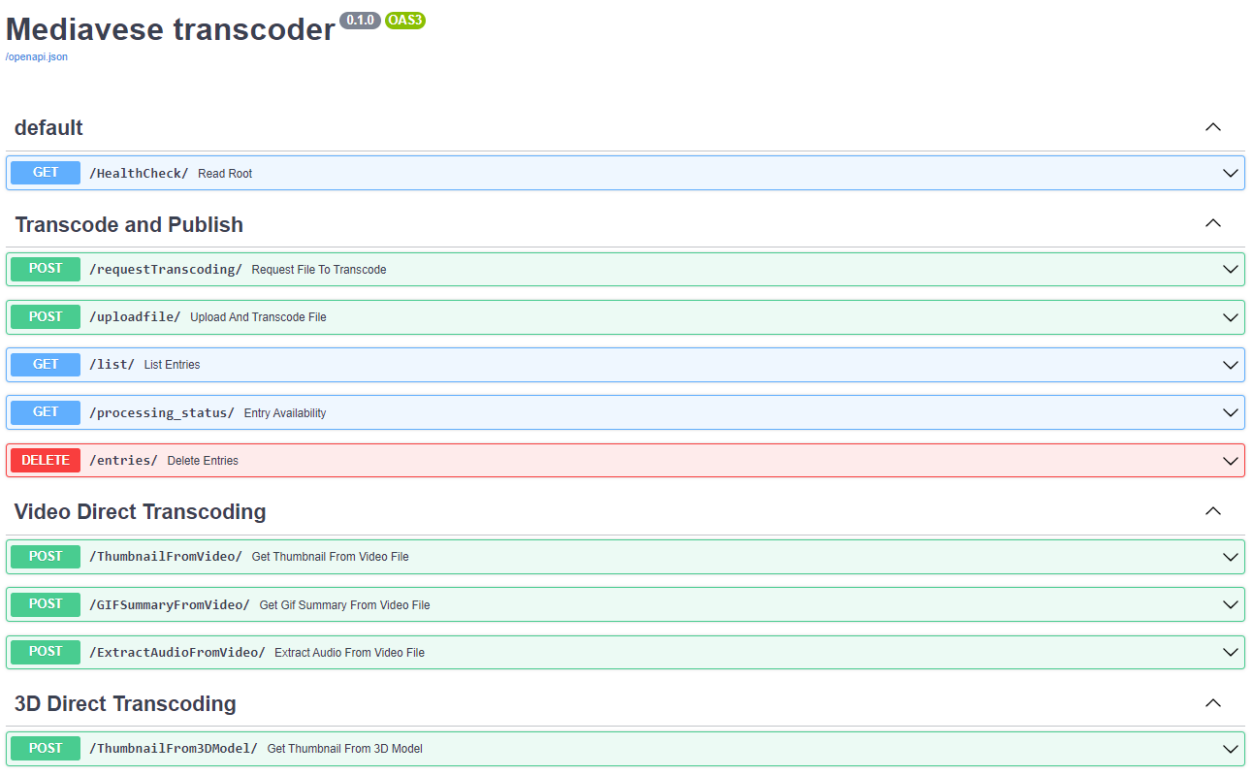


Figure 53: Transcoder Swagger Interface.

5.3.2 IPFS as Media Asset Storage

IPFS is one of the cutting-edge technologies to build web 3.0, it offers the ability to store files in a decentralized way, and it establishes possibilities for creating private networks and provides communication protocols between nodes. Although there is no version tested in production environments yet, the development is very advanced and many of the functionalities are already mature enough to rely on this technology.

IPFS works as follows: When a user adds a file to IPFS, the file is split into smaller chunks, cryptographically hashed, and given a unique fingerprint called an asset identifier (CID). This CID acts as a permanent record of the user's file as it exists at that point in time. When other nodes look up the file, they ask their peer nodes who's storing the asset referenced by the file's CID. When these other nodes view or download the file, they cache a copy — and become another provider of the asset file until their cache is cleared.[34] These features made us think of IPFS as a perfect candidate for sharing single file representations of assets between nodes. Initial satisfactory tests have been carried out for the integration of this technology with the transcoder (video upload, player reproduction), but there is not yet a definitive solution integrated with the platform.

5.4 Asset Adaptation

Some of the services for asset adaptation have been designed to be delivered from a single container. For others it makes more sense to treat them differently. This section will explain in detail which implementations have been tested for each type of asset, the achievements reached, and the possibilities and next steps that have been discovered looking at the final version of the platform.

[34] IPFS Powers the Distributed Web. (2022). Retrieved 26 May 2022, from https://ipfs.io/#how

5.4.1 Text Files

Although text files do not seem to be a very relevant component for a media platform, they will serve different purposes within MediaVerse. Therefore, it has been necessary to create functionalities for them. Three types of text files have been considered as possible to create an asset:

- **.srt:** Will be the main file extension to store subtitles and then associate them to the corresponding videos
- **.json and .xml:** Will be used by external authoring tools to save the configuration of a project and Link a Media verse project to them. Also provide the platform with a tool to save/search/edit systems configurations and share them with related professionals.

In this context, the processing of these types of files will consist in the integration of a simple text editor in the interface, compatible with these three types of files.

5.4.2 Audio

Although there are no specific compression tasks specifically designed for audio files within the project, audio files are part of the framework that are used daily in professional media environments. Therefore, the platform will have to offer the possibility to store and provide audio assets. Consequently, the transcoding service will offer an endpoint with the ability to include automatically the platform's own audio watermark to show the assets to non-owner users.

Based on the research of the different codecs available, it was decided that OPUS would be the one used for this purpose, since it is currently compatible with all modern browsers, it is an open-source alternative, and it has the best quality/bandwidth ratio compared to less innovative codecs (Figure 54).

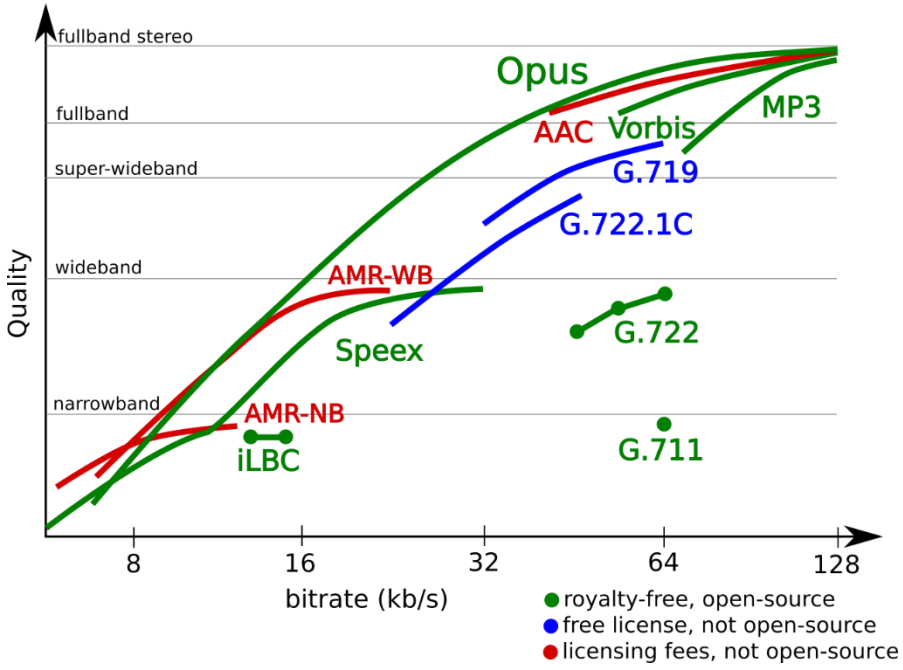


Figure 54: Opus codec quality comparison.

5.4.3 Video

Video optimization fundamentals and implementation examples

All video assets we consume come in package files called containers (Figure 55). Containers act as a wrapper for several streams: video, audio, subtitles, metadata each one of them compressed using different codecs.

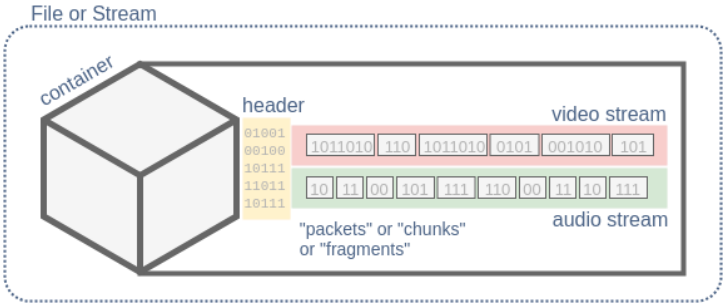


Figure 55: Container format.

Figure 56 represents a real example of the streams and metadata contained in an mp4 file obtained using the ffprobe tool used in the transcoder service:

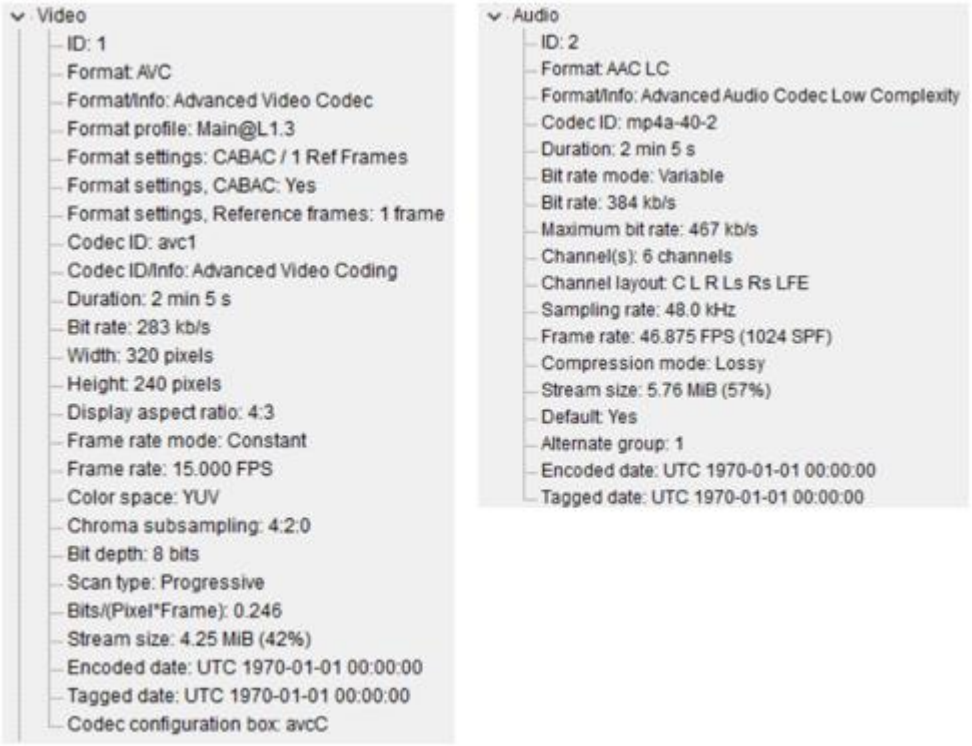


Figure 56: Metadata extracted from a video container.

As the potential users of MediaVerse are professional users, it is expected that the video files they upload to the platform will be of high quality and length. This will result in large container files that cannot be digested by all types of device or viewed quickly. This is where the necessity of using video transcoding arises to enable adaptive streaming.

Video transcoding consists of subjecting the files to four different processes (Figure 57)[35]:

1. Transcoding: changing the video or audio codec of the internal streams.
2. Transmuxing: changing the container format.
3. Transrating: changing the bitrate using different profiles of the same codec.
4. Transsizing: creating several versions for different screen resolutions.

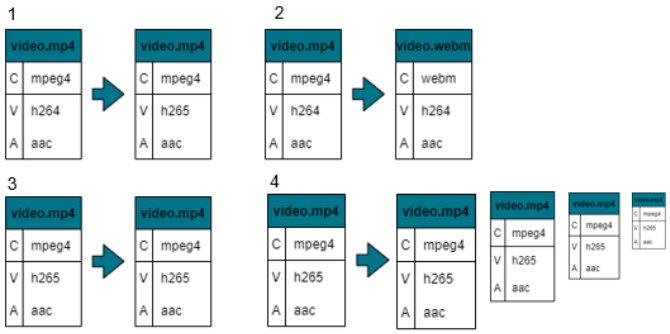


Figure 57: Transcoding processes.

Once the video files have the proper specifications, they are divided into pieces of a few seconds called chunks, to be optimally transmitted over different network capabilities and client devices (Figure 58). All the information related with the location, timing and specifications of the streams contained in these chunks is collected in a manifest file. These manifest files are what media players digest to play the adaptive streaming. The two main protocols used in the industry for this purpose are: HLS and DASH

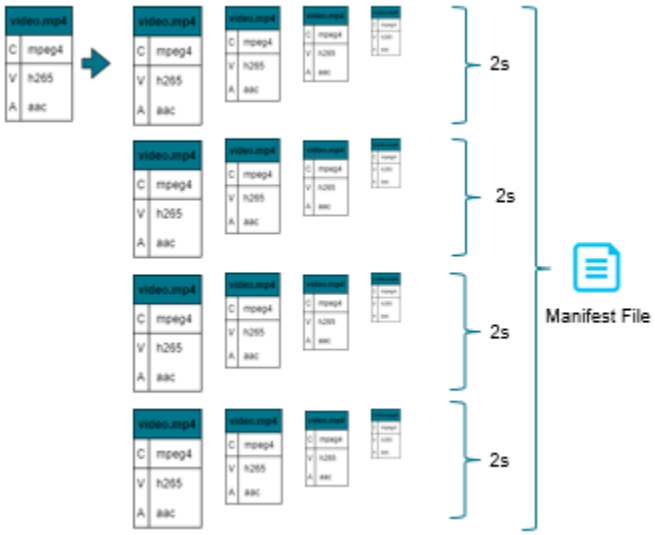


Figure 58: Chunks and Manifest file creation.

The first version of the transcoder uses HLS as it was more convenient to implement a solution compatible with all types of devices. However, testing is also being done using DASH, as it offers more compatibility in certain aspects with the modern codecs we want to use. Once both solutions have tested, we will choose the one that best responds in the context of the player selected for the interface (video.js).

[35] GitHub - leandromoreira/ffmpeg-libav-tutorial: FFmpeg libav tutorial - learn how media works from basic to transmuxing, transcoding and more, 2022

One of the most important decisions that had to be made regarding video quality was the Bitrate and resolution for each of the video qualities offered in the interface. To decide on this, video file manifest files were collected from the main telecom operators in Spain: Orange, Vodafone, from some CDN providers: Kaltura, and compared with the qualities offered by some of the most known video streaming providers: Prime video, Netflix, Youtube.

Manifest files provided by YouTube are not provided in clear from the platform to customers. Therefore, to extract this information we used a script created using pytube[36]. Table 26 shows a comparison of what YouTube is doing for a standard video, and a very popular video.

Table 26: YouTube popular video Adaptive Streaming Qualities.

TYPE	CONTAINER	RESOLUTION	FPS	VIDEO CODEC	AUDIO CODEC	BITRATE (MBPS)
AUDIO	mp4	None	None	None	mp4a.40.5	0,0501
AUDIO	webm	None	None	None	opus	0,056317
VIDEO	mp4	144p	25	av01.0.00M.08	None	0,0745
AUDIO	webm	None	None	None	opus	0,07594
VIDEO	3gpp	144p	6	mp4v.20.3	mp4a.40.2	0,0825
VIDEO	mp4	144p	25	avc1.4d400c	None	0,0957
VIDEO	webm	144p	25	vp9	None	0,096653
AUDIO	mp4	None	None	None	mp4a.40.2	0,1306
AUDIO	webm	None	None	None	opus	0,151267
VIDEO	mp4	240p	25	av01.0.00M.08	None	0,1622
VIDEO	webm	240p	25	vp9	None	0,215869
VIDEO	mp4	240p	25	avc1.4d4015	None	0,2221
VIDEO	mp4	360p	25	av01.0.01M.08	None	0,339651
VIDEO	webm	360p	25	vp9	None	0,396523
VIDEO	mp4	360p	25	avc1.4d401e	None	0,4122
VIDEO	mp4	360p	25	avc1.42001E	mp4a.40.2	0,5007
VIDEO	mp4	720p	25	avc1.64001F	mp4a.40.2	0,5770
VIDEO	mp4	480p	25	avc1.4d401e	None	0,5940
VIDEO	mp4	480p	25	av01.0.04M.08	None	0,609982

[36] pytube — pytube 12.1.0 documentation. (2022). Retrieved 26 May 2022, from https://pytube.io/en/latest/

VIDEO	webm	480p	25	vp9	None	0,668969
VIDEO	mp4	720p	25	avc1.4d401f	None	1,0264
VIDEO	mp4	720p	25	av01.0.05M.08	None	1,226444
VIDEO	webm	720p	25	vp9	None	1,311463
VIDEO	mp4	1080p	25	av01.0.08M.08	None	2,250562
VIDEO	webm	1080p	25	vp9	None	2,646248
VIDEO	mp4	1080p	25	avc1.640028	None	2,9915

The most remarkable features that YouTube has chosen for a not too popular video are: Provide both versions of containers with embedded audio + video and versions with separate audio and video. Use mainly codecs with high compatibility: .h264 and .mp4. Moreover, for a popular video, YouTube spends more computing power to generate many more versions using more modern codecs. VP9, AVC1

The data for the rest of the suppliers analyzed will not be shown here, but in general the approach is the same. Adds several versions, with the same codec, generally h264 due to compatibility.

Given the data analyzed above, it was decided that the objective for the video qualities to be offered by the transcoder would be as follows in Table 27:

Table 27: Target qualities for the transcoder service.

TYPE	CONTAINER	RESOLUTION	FPS	VIDEO CODEC	AUDIO CODEC	BITRATE (MBPS)
AUDIO	mp4	None	None	None	mp4a.40.5	0,0501
AUDIO	mp4	None	None	None	mp4a.40.2	0,1306
VIDEO	mp4	240p	24	av01.0.00M.08	None	0,1364
AUDIO	webm	None	None	None	opus	0,1438
VIDEO	webm	240p	24	vp9	None	0,1685
VIDEO	mp4	240p	24	avc1.4d400d	None	0,1894
VIDEO	webm	360p	24	vp9	None	0,3087
VIDEO	mp4	360p	24	avc1.4d4015	None	0,4759
VIDEO	mp4	480p	24	av01.0.04M.08	None	0,5304
VIDEO	mp4	360p	24	avc1.42001E	mp4a.40.2	0,5507
VIDEO	webm	480p	24	vp9	None	0,5684

VIDEO	webm	720p	24	vp9	None	1,1336
VIDEO	mp4	1080p	24	av01.0.08M.08	None	1,7207
VIDEO	mp4	720p	24	avc1.64001F	mp4a.40.2	1,7396
VIDEO	mp4	720p	24	avc1.4d401f	None	1,8292
VIDEO	webm	1080p	24	vp9	None	2,0172
VIDEO	mp4	1080p	24	avc1.640028	None	3,2762
VIDEO	webm	1440p	24	vp9	None	6,7398
VIDEO	mp4	1440p	24	av01.0.12M.08	None	6,0961
VIDEO	webm	2160p	24	vp9	None	13,3851
VIDEO	mp4	2160p	24	av01.0.12M.08	None	13,4839

Figure 59 shows an example of the output of the transcoding service, master manifest files and the video chunks generated by the MediaVerse video transcoder for a 4K Input file.

Figure 59: transcoder service HLS Output.

Figure 60 shows an example of the asset of the stream manifest files and the .ts files.

```
#EXTM3U
#EXT-X-VERSION:6
#EXT-X-TARGETDURATION:3
#EXT-X-MEDIA-SEQUENCE:0
#EXT-X-PLAYLIST-TYPE:VOD
#EXT-X-INDEPENDENT-SEGMENTS
#EXTINF:3.200000,
stream_9/data00.ts
#EXTINF:1.600000,
stream_9/data01.ts
#EXTINF:1.600000,
stream_9/data02.ts
#EXTINF:1.600000,
stream_9/data03.ts
#EXTINF:2.666667,
stream_9/data04.ts
#EXT-X-ENDLIST
```

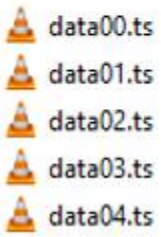


Figure 60: transcoder service chunks Output.

Figure 61, where the asset is tested directly on the platform, depicts the result of this processing. Versions with very good visual quality and for all types of devices and network connections are available.

Figure 61: MediaVerse Player reproducing an Adaptive Streaming Video Created with the transcoder service.

5.4.4 360 Video

There are two approaches in the project for processing 360-degree video. The first one aims to support flat 360-degree video by the implemented transcoding service, also creating accessible versions for all types of devices.

It has been possible to create a video pipeline like the one described above using different qualities and configurations, adapted to 360-degree videos, with streams ranging from 3Mbps to 30Mbps. The problem with this approach is that unless the network quality is excellent, using for example a 5G infrastructure, the network will not be able to support multiple users playing 3Mbps to 30Mbps streams.

In this context, the second approach to 360-degree video is proposed, involving the cutting edge OMAF specification to solve this problem and generate 360-degree video assets rendered from the server and optimized by dividing the 360-degree spheres into several tiles.

5.4.5 3D Models

Supporting 3D assets is not one of the usual tasks of a transcoding server. However, since this project intends to integrate all kinds of assets, the first steps have been taken to be able to generate useful transformations for the interface also from this module. In this case, an endpoint has been developed to generate Thumbnails through 3D models. This service intends to evolve to be able to show a rotating GIF file from a model.

To this end, different challenges had to be overcome:

- There were no native tools that provided this service automatically with an external library.
- Software libraries dedicated to model processing require a graphical interface.
- The position of the camera, the3D model and the lights must be automatically calculated so that very different types of 3D mesh are displayed on the screen.

To solve these problems, the library Panda3D was chosen, which is an open-source, free-to-use 3D engine made for the development of Realtime 3D games, visualizations, simulations, experiments. It consists of a library of subroutines that can be used with either the Python or C++ programming language, as well as a few tools to assist with development and debugging.[37] The models are rendered using Panda3D on a virtual framebuffer X server, installed on the CentOS host: XVFF[38]. Finally, the XVFB wrapper: pyvirtualdisplay[39] has been used to run the virtual window from the Python code. Figure 62 shows the result of the transformation, presenting a real example, in which a relatively complex 3DModel has been processed, and is correctly textured and illuminated.

Figure 62: Transcoder Thumbnail created from a .glb 3DModel.

[37] Panda3D Manual — Panda3D Manual. (2022). Retrieved 26 May 2022, from https://docs.panda3d.org/1.10/python/index
[38] XVFB. (2022). Retrieved 26 May 2022, from https://www.x.org/releases/X11R7.6/doc/man/man1/Xvfb.1.xhtml
[39] GitHub - ponty/PyVirtualDisplay at 3.0. (2022). Retrieved 26 May 2022, from https://github.com/ponty/pyvirtualdisplay/tree/3.0

5.5 Improved 360-degree Transcoding

Videos and images produced with 360-degree cameras can result in different video and image formats and projections, depending on the type of camera and methods to merge the individual camera captures (the process is also known as stitching). While the scope of MediaVerse excludes stitching as a provided adaptation service, we want to give future users the opportunity to use their uploaded high-quality 360-degree video for web-based consumer facing products by implementing an advanced method for adaptive streaming that enables high quality output while optimizing used bandwidth.

Since a 360-degree camera captures the whole sphere, it also needs to be presented to the viewers in full dimensions and resolutions. This requires high throughput networks, as for simple streaming in equirectangular formats, the base resolution needs to be at least 1440p to provide an adequate output resolution for an average field of view (FoV) of 60 degrees. The corresponding horizontal resolution would be 240 pixels at the visual equator in that case.

ISO/IEC 23090-2 Omnidirectional Media Format (OMAF): is an MPEG standard for the storage and delivery format of 360-degree content (Figure 63). It features a tile-based High Efficiency Video Coding (HEVC) approach. An equirectangular or cube map projection 360-degree video is further split into viewport-dependent tiles of different resolution. This ensures that the main view direction is streamed in a high resolution without wasting too much bandwidth on the non-visible parts.

- 5 second timelapse
- 5760x2880 pixel
- 112.9 MB
- Raw video file 3.73GB

1. avmediainfo or ffprobe for initial parameters

2.
```
ffprobe -select_streams v:0 -show_entries stream=nb_frames \
-of default=nokey=1:noprint_wrappers=1 -i sample/car_race2.yuv \
-video_size 1920x960 -f rawvideo
```

3.
```
ffmpeg -r 24 -i ./../sample/DumboTimelapse_5760x2880.mp4 \
-c:v rawvideo -pixel_format yuv420p \
./../sample/DumboTimelapse_5760x2880_24fps.yuv
```

4.
```
./create_omaf_files.py -s 1-5 \
-i ./../sample/DumboTimelapse_5760x2880_24fps.yuv \
-wdt 5760 -hgt 2880 -f 149 -fr 24 -q 25 -t 6 --codec 1 \
-o kvazaarEncoding
```

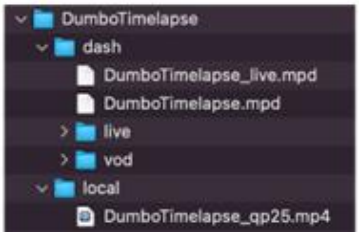


Figure 63: Parameters for FoVeated optimisation and segmentation of 360-degree content using OMAF.

5.5.1 Transformation-as-client Architecture

There are several options to store and access intermediate and final results from adaptations. One approach is to design services that inversely operate on the DAM API only to request original media and provide adaptations.

Integration

The design approach for the MediaVerse OMAF transcoder is essentially a wrapper for configuration and file management around an existing implementation for the actual worker. The worker is the experimental OMAF

file creation tools implementation[40] by Fraunhofer HHI[41]. It creates 24 viewport streams that are streamed based on a viewers' quaternion[42].

A user wanting to use the MediaVerse OMAF transcoder can log in with their credentials from an associated (or selectable) MV node. This allows for optimal flexibility, as the computing intensive OMAF transcoder can cater to multiple MV nodes. Once a user is authenticated, they can query MV node projects or suitable assets for being transcoded. An analyser component obtains data from the input file that is needed to start the OMAF conversion with suitable parameters, such as a total number of raw frames available for a conversion. Temporary files and the result are kept on a local storage, as computing performance and storage volume requirements are high.

The output result is a manifest file and a multi-level folder structure containing all the encoded video fragments. As such, it cannot be simply uploaded as a single asset into the DAM. A solution is to upload a manifest file as a text file into the DAM. This text file contains information about the manifest URL, its data type, and additional information. It should be considered for future improvements, that there should be an asset type available to cater to such tools that are producing such multi-file output of a transformation.

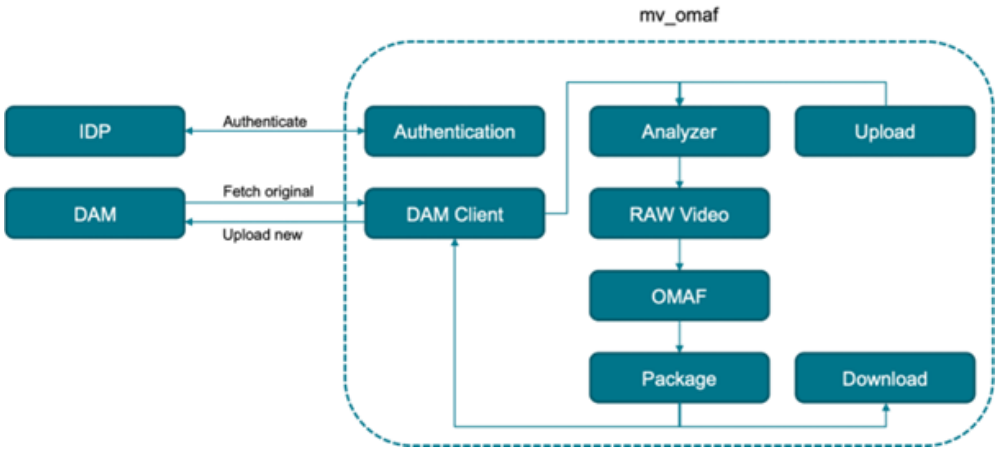


Figure 64: Overview of a transformation client using the OMAF transcoder as an example.

Internal Architecture

The MediaVerse OMAF transcoding service consists of a Phoenix/Elixir web application and a PostgreSQL database provided as a containerized service using Docker. It can be deployed standalone as a service to produce OMAF compliant adaptive streaming 360-degree video using GitHub and Google as authentication methods (can be extended to other OAuth2 compliant IDPs), as well as allowing to log in using a specified MediaVerse node.

Container: The transcoding service is based on a headless (non-GUI) Debian image, as it offers all necessary tools and libraries compatible to the Elixir/Phoenix versions we are using. Besides common packages for building and running the applications, most notably `ffmpeg` is needed to support the video analysis. The OMAF file creation tool requires Python 2.7 and several encoder and projection tools that come as compiled static binaries as a dependency of the MediaVerse OMAF transcoder repository.

[40] https://github.com/fraunhoferhhi/omaf.js/tree/master/omaf-file-creation
[41] https://github.com/fraunhoferhhi/omaf.js
[42] https://en.wikipedia.org/wiki/Quaternion

Elixir/Phoenix: Elixir is a dynamic, functional language for building scalable and maintainable applications[43]. We are using the current version 1.13. Phoenix is a web framework for the Elixir programming language based on the common MVC paradigm and is used in version 1.6.6. It uses a database to persist information on users, files, and jobs. All routes are protected and require a logged in account. The application consists of several modules described in the following paragraphs.

DAM client: The DAM client is a module is used to interface with a MediaVerse node, query projects and assets and upload the manifest files for the result of a transformation.

Analyser module: The analyser module is a wrapper for `ffmpeg`/`ffprobe` to obtain parameters for further processing. Since the video is transformed into raw video, it is crucial to obtain the number of frames available from the compressed video. This is achieved through calling `ffprobe` or `avmediainfo` commands. Using the obtained framerate, one can use `ffmpeg` to convert the video source into a raw video (YUV) file:

Worker module: With the dimension parameters set and the numbers of frames available, the OMAF file creation tool (using a python wrapper) can be called by the worker module. The worker module is a wrapper for the OMAF file creation tool script, written in Python. It uses parameters obtained by the analyser that are required.

[43] https://elixir-lang.org/

6 Conclusion

This deliverable reported on the output of research and development tasks focusing on the intelligent analysis, retrieval and management of MediaVerse assets.

The media annotation tool can produce accurate text annotation for any kind of media. The annotations not only describe objects present in the contents but also the actions happening, and the faces depicted if those correspond to celebrities. This maps content into the same space where the eventual query for looking for content is made, i.e., the text space, where the problem is reduced to a text search. Image annotation has been performed on tasks including Meme detection, Action recognition, Disturbing content detection. MemeTector achieved superior performances compared to fine-tuned classification models as ViT, EfficientNetB5. ResNet152 has been trained on Kinects400 to perform action recognition on static images. For Video annotation, the work primarily focused on Action recognition. Though TimeSformer was initially chosen as the best model (in terms of produced tags and time consumption), it raised ONNX deployment issues, therefore Slowfast was chosen. To provide the video with further useful information, temporal segmentation methods were used to select key frames and apply image annotation methods on them. Given the difficulties associated to 3D content annotation task, it was split in two sub tasks: 3D object annotation and 3D scene annotation. For each of these, a Multiview approach has been employed taking advantage of ANN already deployed for image annotation. Because of the absence of a suitable dataset, results are presented visually in a qualitative fashion. The proposed approach is able to detect concepts in the scene. Several services have been evaluated to enable non-AI-experts to build their own models for niche tasks pertinent to their work. In terms of graphics and analytics provided to the user, the directions of DeepVA and Tensorflow's Deep Playground seem appealing. The media annotation service exposes a gRPC API which has, in addition to the bidirectional streaming endpoint discussed in D3.1, three unary RPC, ImageAnnotation, ThreeDAnnotation and VideoAnnotation, responsible for annotating images, 3D assets and videos respectively.

The Cross-Modal Retrieval system leverages the CLIP large scale model and Faiss to speed up similarity search of both retrieval and recommendation systems. CLIP encoders are trained to embed asset from different media types into a joint semantic space where they can be directly compared. This allows us to conduct extensive evaluation of our CMR system in terms of similarity measures (cosine, L2-norm, dot-product). Our experiments are conducted on MSCOCO validation split of 2014 where CLIP achieves R@10 values of 0.889 for the tasks of Image retrieval and 0.910 on text retrieval. The complete CMR system, which also includes the fast vector search module Faiss, is able to retain these values of recall while speeding up the search process from 129.31 up to 0.72 ms (searching in a database of 5000 assets). Since the module is designed to deal with texts and images, satisfying both cross-modality and uni-modality searches, it can very well be used in conjunction with the annotation services (described in section 2) which generate semantic textual tags for different data types that can, subsequently, be used as inputs of CLIP text encoder. The CMR system is by default designed to return a fixed number of results in response to his query, however, a more advanced selection criterion is explored in Section 3.6, where the number of results is decided dynamically based on relevance of the asset to the query. However, as CLIP is likely to be applied in a wide variety of scenarios the proposed similarity thresholds must be validated over a larger number of datasets.

The recommendation system leverages the available knowledge about such as the asset that has been of interest in the past (for example by what has been searched or posted by the user). This asset, both texts and images, represented in a multidimensional space obtained with the CLIP encoder are then grouped with HDBSCAN technique each time a recommendation is requested, and then a maximum of three seeds are selected, therefore

limiting the subjects of the output (and thus the diversity). The system has been studied utilizing diversity as performance indicators. Experiments have been run on a monitored environment, which allowed us to map user posted contents to macro topics, each identified as a theoretical cluster against which we expect consistency from the recommender system. The user seeds, generated from previous contents posted by the user, have been proved to be - qualitatively – able to retrieve contents semantically related to user interests. A quantitative analysis has been performed on the system capability to return a certain degree of diversity among recommended contents. Given the parametric nature of the diversity, the recommendation system controls the diversity by changing the fraction of recommended asset returned from a random outlier. It was shown that the parameters of the system have limited influence on the range of values of the diversity that remains between 0.80 and 0.99.

Finally, the Content Adaptation framework supports the ingestion, manipulation, and sharing of the content for all media type supported optimizing the data format so that both the most modern networks technologies, i.e., 5G, can be fully taken advantage of and the content can be delivered in a real-time fashion.

7 References

Adomavicius, G., & Tuzhilin, A. (2005). Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. IEEE transactions on knowledge and data engineering, 17(6), 734-749.

Agarwal, N., Haque, E., Liu, H., & Parsons, L. (2005). Research paper recommender systems: A subspace clustering approach. In International Conference on Web-Age Information Management (pp. 475-491). Springer.

Andrew, G., Arora, R., Bilmes, J., & Livescu, K. (2013). Deep canonical correlation analysis. In International conference on machine learning, 1247-1255, PMLR.

Antikacioglu, A., Bajpai, T., & Ravi, R. (2019). A new system-wide diversity measure for recommendations with efficient algorithms. SIAM Journal on Mathematics of Data Science, 1(4), 759-779.

Armeni, I., Sener, O., Zamir, A. R., Jiang, H., Brilakis, I., Fischer, M., & Savarese, S. (2016). 3d semantic parsing of large-scale indoor spaces. In Proceedings of the CVPR, 1534-1543.

Arnab, A., Dehghani, M., Heigold, G., Sun, C., Lucic, M. & Schmid, C. (2021). ViViT: A Video Vision Transformer. In Proceedings of ICCV.

Bertasius, G., Wang, H., & Torresani, L. (2021). Is Space-Time Attention All You Need for Video Understanding? ArXiv, abs/2102.05095.

Bizer, C., Lehmann, J., Kobilarov, G., Auer, S., Becker, C., Cyganiak, R., & Hellmann, S. (2009). Dbpedia-a crystallization point for the web of data. Journal of web semantics, 7(3), 154-165.

Boser, B. E., Guyon, I. M., & Vapnik, V. N. (1992). A training algorithm for optimal margin classifiers. In Proceedings of the fifth annual workshop on Computational learning theory, 144-152.

Burke, R. (2000). Knowledge-based recommender systems. Encyclopedia of library and information systems, 69 (Supplement 32), 175-186.

Cao, Q. , Shen, L., Xie, W., Parkhi, O. & Zisserman, A. (2018). VGGFace2: A Dataset for Recognising Faces across Pose and Age, in 2018 13th IEEE International Conference on Automatic Face & Gesture Recognition (FG 2018), Xi'an, China, pp. 67-74.

Carreira, J. & Zisserman, A. (2017). Quo Vadis, Action Recognition? A New Model and the Kinetics Dataset. Proceedings of the CVPR, 4724-4733.

Chen, H., Ding, G., Liu, X., Lin, Z., Liu, J., & Han, J. (2020). Imram: Iterative matching with recurrent attention memory for cross-modal image-text retrieval. In Proceedings of the CVPR, 12655-12663.

Chen, T., Kornblith, S., Norouzi, M., & Hinton, G. (2020). A simple framework for contrastive learning of visual representations. In International conference on machine learning, 1597-1607, PMLR.

Chen, W. Y., Liu, Y. C., Kira, Z., Wang, Y. C. F., & Huang, J. B. (2018, September). A Closer Look at Few-shot Classification. In International Conference on Learning Representations.Chen, Y. C., Li, L., Yu, L., El Kholy, A., Ahmed, F., Gan, Z., ... & Liu, J. (2020). Uniter: Universal image-text representation learning. In European conference on computer vision (pp. 104-120). Springer, Cham.

Chen, Y., Liu, Z., Xu, H., Darrell, T. & Wang, X. (2021). Meta-Baseline: Exploring Simple Meta-Learning for Few-Shot Learning, ICCV.

Chia, P. J., Attanasio, G., Bianchi, F., Terragni, S., Magalhães, A. R., Goncalves, D., ... & Tagliabue, J. (2022b). Fashionclip: Connecting language and images for product representations. arXiv preprint arXiv:2204.03972.

Chia, P. J., Tagliabue, J., Bianchi, F., Greco, C., & Goncalves, D. (2022a). " Does it come in black?" CLIP-like models are zero-shot recommenders. arXiv preprint arXiv:2204.02473.

Dai, A., & Nießner, M. (2018). 3dmv: Joint 3d-multi-view prediction for 3d semantic scene segmentation. In Proceedings of the European Conference on Computer Vision (ECCV), 452-468.

Dai, A., Chang, A. X., Savva, M., Halber, M., Funkhouser, T., & Nießner, M. (2017). Scannet: Richly-annotated 3d reconstructions of indoor scenes. In Proceedings of the CVPR, 5828-5839.

Diao, H., Zhang, Y., Ma, L., & Lu, H. (2021). Similarity reasoning and filtration for image-text matching. In Proceedings of the AAAI Conference on Artificial Intelligence 35(2), 1218-1226.

Doersch, C., Gupta, A., & Efros, A. A. (2015). Unsupervised visual representation learning by context prediction. In Proceedings of the CVPR, 1422-1430.

Donahue, J., & Simonyan, K. (2019). Large scale adversarial representation learning. Advances in neural information processing systems, 32.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., ... & Houlsby, N. (2020). An image is worth 16x16 words: Transformers for image recognition at scale. arXiv preprint arXiv:2010.11929.

Faghri, F., Fleet, D. J., Kiros, J. R., & Fidler, S. (2017). Vse++: Improving visual-semantic embeddings with hard negatives. arXiv preprint arXiv:1707.05612.

Feichtenhofer, C. (2020). X3D: Expanding Architectures for Efficient Video Recognition. CVPR.

Feichtenhofer, C., Fan, H., Malik, J. & He, K. (2019). SlowFast Networks for Video Recognition. ICCV.

Feng, F., Wang, X., & Li, R. (2014). Cross-modal retrieval with correspondence autoencoder. In Proceedings of the 22nd ACM international conference on Multimedia (pp. 7-16).

Grill, J. B., Strub, F., Altché, F., Tallec, C., Richemond, P., Buchatskaya, E., ... & Valko, M. (2020). Bootstrap your own latent-a new approach to self-supervised learning. Advances in neural information processing systems, 33, 21271-21284.

Guo, Y., Wang, H., Hu, Q., Liu, H., Liu, L., & Bennamoun, M. (2020). Deep learning for 3d point clouds: A survey. IEEE transactions on pattern analysis and machine intelligence, 43(12), 4338-4364.

Gurari, D., Zhao, Y., Zhang, M., & Bhattacharya, N. (2020). Captioning images taken by people who are blind. In European Conference on Computer Vision, 417-434. Springer, Cham.

Hardoon, D. R., Szedmak, S., & Shawe-Taylor, J. (2004). Canonical correlation analysis: An overview with application to learning methods. Neural computation, 16(12), 2639-2664.

He, J., Guo, Z., Shao, Z., Zhao, J., & Dan, G. (2020). An LSTM-based prediction method for lower limb intention perception by integrative analysis of Kinect visual signal. Journal of healthcare engineering, 2020.

He, K., Fan, H., Wu, Y., Xie, S., & Girshick, R. (2020). Momentum contrast for unsupervised visual representation learning. In Proceedings of the CVPR, 9729-9738.

Hodosh, M., Young, P., & Hockenmaier, J. (2013). Framing image description as a ranking task: Data, models and evaluation metrics. Journal of Artificial Intelligence Research, 47, 853-899.

Jannach, D., Zanker, M., Ge, M., & Gröning, M. (2012). Recommender systems in computer science and information systems–a landscape of research. In International conference on electronic commerce and web technologies, 76-87. Springer, Berlin, Heidelberg.

Jaritz, M., Gu, J., & Su, H. (2019). Multi-view pointnet for 3d scene understanding. In Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops.

Ji, S., Xu, W., Yang, M. & Yu, K. (2013). 3D Convolutional Neural Networks for Human Action Recognition. In IEEE Transactions on Pattern Analysis and Machine Intelligence, 35(1), 221-231.

Jia, C., Yang, Y., Xia, Y., Chen, Y. T., Parekh, Z., Pham, H., ... & Duerig, T. (2021). Scaling up visual and vision-language representation learning with noisy text supervision. In International Conference on Machine Learning, 4904-4916. PMLR.

Karpathy, A., Toderici, G., Shetty, S., Leung, T., Sukthankar, R. & Fei-Fei, L. (2014). Large-Scale Video Classification with Convolutional Neural Networks. In Proceedings of the CVPR, 1725-1732.

Kay, W., Carreira, J., Simonyan, K., Zhang, B., Hillier, C., Vijayanarasimhan, S., Viola, F., Green, T., Back, T., Natsev, P., Suleyman, M. & Zisserman, A. (2017). The Kinetics Human Action Video Dataset. arXiv:1705.06950.

Kingma, D. P., Mohamed, S., Jimenez Rezende, D., & Welling, M. (2014). Semi-supervised learning with deep generative models. Advances in neural information processing systems, 27.

Kiros, R., Salakhutdinov, R., & Zemel, R. S. (2014). Unifying visual-semantic embeddings with multimodal neural language models. arXiv preprint arXiv:1411.2539.

Kiros, R., Zhu, Y., Salakhutdinov, R. R., Zemel, R., Urtasun, R., Torralba, A., & Fidler, S. (2015). Skip-thought vectors. Advances in neural information processing systems, 28.

Kolesnikov, A. et al. (2020). Big Transfer (BiT): General Visual Representation Learning. In: Vedaldi, A., Bischof, H., Brox, T., Frahm, JM. (eds) Computer Vision – ECCV 2020, vol 12350. Springer, Cham.

Krishna, R., Zhu, Y., Groth, O., Johnson, J., Hata, K., Kravitz, J., ... & Fei-Fei, L. (2017). Visual genome: Connecting language and vision using crowdsourced dense image annotations. IJCV, 123(1), 32-73.

Kundu, A., Yin, X., Fathi, A., Ross, D., Brewington, B., Funkhouser, T., & Pantofaru, C. (2020). Virtual multi-view fusion for 3d semantic segmentation. In European Conference on Computer Vision, 518-535, Springer, Cham.

Kużelewska, U., & Wichowski, K. (2015). A modified clustering algorithm DBSCAN used in a collaborative filtering recommender system for music recommendation. In International conference on dependability and complex systems, 245-254, Springer, Cham.

Lee, K. H., Chen, X., Hua, G., Hu, H., & He, X. (2018). Stacked cross attention for image-text matching. In Proceedings of the European conference on computer vision (ECCV), 201-216.

Li, X., Yin, X., Li, C., Zhang, P., Hu, X., Zhang, L., ... & Gao, J. (2020). Oscar: Object-semantics aligned pre-training for vision-language tasks. In European Conference on Computer Vision, 121-137. Springer, Cham.

Lin, T. Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., ... & Zitnick, C. L. (2014). Microsoft coco: Common objects in context. In European conference on computer vision, 740-755. Springer, Cham.

Liu, Z., Lin, Y., Cao, Y., Hu, H., Wei, Y., Zhang, Z., Lin, S. & Guo, B. (2021a). Swin Transformer: Hierarchical Vision Transformer using Shifted Windows. arXiv:2103.14030.

Liu, Z., Luo, P., Wang, X. & Tang, X. (2015). Deep Learning Face Attributes in the Wild, ICCV.

Liu, Z., Ning, J., Cao, Y., Wei, Y., Zhang, Z., Lin, S. & Hu, H. (2021b). Video Swin Transformer. arXiv:2106.13230.

Lops, P., Jannach, D., Musto, C., Bogers, T., & Koolen, M. (2019). Trends in content-based recommendation. User Modeling and User-Adapted Interaction, 29(2), 239-249.

Lu, J., Batra, D., Parikh, D., & Lee, S. (2019). Vilbert: Pretraining task-agnostic visiolinguistic representations for vision-and-language tasks. Advances in neural information processing systems, 32.

Mladenic, D. (1999). Text-learning and related intelligent agents: a survey. IEEE intelligent systems and their applications, 14(4), 44-54.

Monti, D. M. (2020). Multicriteria Evaluation for Top-k and Sequence-based Recommender Systems.

Mori, S., Crain, B. J., Chacko, V. P., & Van Zijl, P. C. (1999). Three-dimensional tracking of axonal projections in the brain by magnetic resonance imaging. Annals of Neurology: Official Journal of the American Neurological Association and the Child Neurology Society, 45(2), 265-269.

Musto, C. (2010). Enhanced vector space models for content-based recommender systems. In Proceedings of the fourth ACM conference on Recommender systems, 361-364.

Nakamura, A. & Harada, T. (2019). Revisiting Fine-tuning for Few-shot Learning, arXiv:1910.00216.

Ngiam, J., Khosla, A., Kim, M., Nam, J., Lee, H., & Ng, A. Y. (2011). Multimodal deep learning. In ICML.

Oord, A. V. D., Li, Y., & Vinyals, O. (2018). Representation learning with contrastive predictive coding. arXiv preprint arXiv:1807.03748.

Peng, Y., Huang, X., & Qi, J. (2016). Cross-media shared representation by hierarchical learning with multiple deep networks. In IJCAI, 3846-3853.

Qiu, S., Anwar, S., & Barnes, N. (2021). Semantic segmentation for real point cloud scenes via bilateral augmentation and adaptive fusion. In Proceedings of the CVPR, 1757-1767.

Radford, A., Kim, J. W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., ... & Sutskever, I. (2021). Learning transferable visual models from natural language supervision. In ICML, 8748-8763. PMLR.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1985). Learning internal representations by error propagation. California Univ San Diego La Jolla Inst for Cognitive Science.

Sharma, C., & Kaul, M. (2020). Self-supervised few-shot learning on point clouds. Advances in Neural Information Processing Systems, 33, 7212-7221.

Simonyan, K. & Zisserman, A. (2014). Two-Stream Convolutional Networks for Action Recognition in Videos. Proceedings of the 27th International Conference on Neural Information Processing Systems, 568–576, 2014.

Su, H., Maji, S., Kalogerakis, E., & Learned-Miller, E. (2015). Multi-view convolutional neural networks for 3d shape recognition. In Proceedings of the IEEE international conference on computer vision, 945-953.

Thomee, B., Shamma, D. A., Friedland, G., Elizalde, B., Ni, K., Poland, D., ... & Li, L. J. (2016). YFCC100M: The new data in multimedia research. Communications of the ACM, 59(2), 64-73.

Tran, D., Bourdev, L., Fergus, R., Torresani, L., & Paluri, M. (2015). Learning spatiotemporal features with 3D convolutional networks. In Proceedings of the IEEE international conference on computer vision, 4489-4497.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. Advances in neural information processing systems, 30.

Wang L. et al. (2016) Temporal Segment Networks: Towards Good Practices for Deep Action Recognition. In: Leibe B., Matas J., Sebe N., Welling M. (eds) Computer Vision – ECCV 2016. vol 9912. Springer, Cham.

Wang, K., He, R., Wang, W., Wang, L., & Tan, T. (2013). Learning coupled feature spaces for cross-modal matching. In Proceedings of the ICCV, 2088-2095.

Wang, K., Yin, Q., Wang, W., Wu, S., & Wang, L. (2016). A comprehensive survey on cross-modal retrieval. arXiv preprint arXiv:1607.06215.

Wei, X., Zhang, T., Li, Y., Zhang, Y., & Wu, F. (2020). Multi-modality cross attention network for image and sentence matching. In Proceedings of the CVPR, 10941-10950.

Yang, Y., Luo, Y., Chen, W., Shen, F., Shao, J., & Shen, H. T. (2016). Zero-shot hashing via transferring supervised knowledge. In Proceedings of the 24th ACM international conference on Multimedia, 1286-1295.

Yang, Y., Zha, Z. J., Gao, Y., Zhu, X., & Chua, T. S. (2014). Exploiting web images for semantic video indexing via robust sample-specific loss. IEEE Transactions on Multimedia, 16(6), 1677-1689.

Ye, J., Chen, Z., Liu, J., & Du, B. (2020, July). TextFuseNet: Scene Text Detection with Richer Fused Features. In IJCAI Vol. 20, 516-522.

Zhai, X., Peng, Y., & Xiao, J. (2013). Learning cross-media joint representation with sparse and semisupervised regularization. IEEE Transactions on Circuits and Systems for Video Technology, 24(6), 965-978.

Zhang, Q., Lei, Z., Zhang, Z., & Li, S. Z. (2020). Context-aware attention network for image-text retrieval. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, 3536-3545.

Zhao, N., Chua, T. S., & Lee, G. H. (2021). Few-shot 3d point cloud semantic segmentation. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 8873-8882.

Zhuang, Y. T., Wang, Y. F., Wu, F., Zhang, Y., & Lu, W. M. (2013). Supervised coupled dictionary learning with group structures for multi-modal retrieval. In Twenty-Seventh AAAI Conference on Artificial Intelligence.

MediaVerse is an H2020 Innovation Project co-financed by the EC under Grant Agreement ID: 957252.