Blueprint for a Production-Ready Information Retrieval System based on Multi-Modal Embeddings

André Ebert[1], Anika Apel[2], Piotr Chodyko[2], Kyle Hiroyasu[2], Festina Ismali[2], Hyein Koo[2], Julia Kronburger[1], Robert Pesch[1]

Abstract: Deep Learning models for mapping documents from different domains, e.g., text, images, and audio, into a common vector space, enable a seamless information retrieval between the different domains and, thus, significantly improve the user experience of many expert tools. Despite various models for multi-modal mappings presented in scientific literature, the implementation and integration remain a challenge within the industry, especially for small or medium-sized companies. Reasons are, that developing such retrieval systems for production use-cases is a non-trivial task, requiring scalable, reliable, and cost-efficient infrastructure, services as well as adequate Deep Learning models. We present a generic and flexible blueprint architecture, targeting the development of a production-ready image-text retrieval search system using *Kubernetes*, *MLflow*, *Elasticsearch*, and integrating state-of-the-art Deep Learning models.

Keywords: Information Retrieval; Image-to-Text; Multi-Modal Embeddings; Deep Learning; Artificial Intelligence; Data-Science to Production

1 Introduction

With the increasing adoption of smart devices and the explosion of available data, methods to search and retrieve information have grown in importance. In the most familiar settings, text-based retrieval systems such as modern-day search engines, allow searching large corpora of documents to identify relevant information. Moreover, research has focused on multi-modal retrieval, which is the task of retrieving information from different forms of media to enable seamless information retrieval from various domains, e.g., texts, images, or audio records. Thus, for example, users of an image-text-retrieval system can search for the most relevant text documents using a photo as query input or vice-versa. Machine Learning and especially Deep Learning models have proven to be a powerful tool for such retrieval systems [Fa18, ZL18, Li20, Wa19, Ch20]. Numerous blog posts demonstrate the simplicity of implementing highly sophisticated models from the literature with millions of parameters using modern Machine Learning frameworks like *PyTorch* and *TensorFlow* in simple notebooks. Unfortunately, the transformation of these prototypes into a data product in the

[1] inovex GmbH, Ludwig-Erhard-Allee 6, 76131 Karlsruhe, Germany
aebert@inovex.de, jkronburger@inovex.de, rpesch@inovex.de
[2] Technische Universität München, Fakultät für Informatik, Boltzmannstr. 3, 85748 Garching, Germany
anika.apel@tum.de, piotr.chodyko@gmail.com, kylehiroyasu.tum@gmail.com, festina.ismali@gmail.com, hyeinkooo@gmail.com

industry proves to be a non-trivial task. Apart from rigorous model evaluation, advanced data engineering, software engineering, and IT-operations know-how are required to address the hidden technical debts in Machine Learning systems [Sc15], e.g., for building and maintaining a serving infrastructure, addressing scalability issues, introducing *Continuous Integration and Continuous Development* (CI/CD) pipelines, including model versioning, dealing with prediction uncertainties, and model as well as system monitoring.

This work describes the implementation of an image-text-retrieval system based on deep multi-modal embeddings and its deployment within a scalable, lean, and robust architecture. In this scenario, an image is used as a query to retrieve text documents with a high relevance. Besides objectives like rapid response times and the quality of results, the system's production readiness, as well as technical details concerning its architecture, are of importance. In this context, the presented system delivers the blueprint for deploying and operating an information retrieval system based on deep multi-modal embeddings, using in-cloud and on-premise based container environments.

2 Deep Learning Approaches for Multi-Modal Information Retrieval

Deep Learning based image-text-retrieval approaches can be categorized based on the way the final cross-modal embeddings are obtained [Ch20]: (i) *Pairwise Learning*, (ii) *Interaction Learning*, and (iii) *Attributes Learning*.

Pairwise Learning methods aim at embedding texts and images in a joint vector space. The goal is to obtain semantically similar vector embeddings for images and texts. These approaches consist commonly of two branches, one for encoding images and another one for encoding texts. The outcomes of both branches are vectors, whose similarity can be calculated. The measurement of their similarity provides a supervisory signal, that is used to improve the encoding during training. Prominent examples of such methods are *VSE++* and *DCMP* [Fa18, ZL18]. In contrast, *Interaction Learning* methods like *Oscar* or *CAMP* [Li20, Wa19] tend to achieve high accuracy rates compared to other approaches. Images and texts are not encoded separately and information flows between both branches before obtaining the embeddings in the joint vector space. This enables learning of correspondences between particular parts of images and texts. However, such interactions make these methods prohibitively expensive in terms of computation. Methods of *Attributes Learning*, such as *VSRN* and *ACMM* [Li19, HW19], obtain high-level feature texts and images to compute correlations between those. Compared to interaction methods there is no flow of information between text and image branches when creating vector representations.

In case of the production-ready system targeted by this work, the integrability in a target architecture and the fast retrieval of decent results is decisive. Thus, the sheer accuracy scores of a model is not the only selection criterion, as described in the following section.

3 Requirements

In order to cover the demands of a highly efficient, generic, and capable platform for multi-modal information retrieval, its fundamental requirements are specified in prior. In order to address specific needs and preconditions as they can be found within small and medium sized businesses, the requirements were derived in collaboration with the consortial partners of the Service-Meister project as follows:

Generalizability: The architecture design allows easy customization to enable the implementation of a diverse range of different applications.
Affordability: Despite the in general demanding features of a state-of-the-art platform relying on Deep Learning, the necessary expenses must scale with the actually used resources and also be affordable for small- and medium-sized businesses.
Scalability: In order to handle a sparse as well as a massive amount of incoming requests, the architecture must be implemented in a scalable way. New resources, such as storage, virtual environments, and processing power are acquired on-demand and without any manual intervention.
Modularity and Extensibility: These aspects cover two crucial challenges: 1) Machine Learning models must be switchable without additional effort in order to deploy such with a better performance in a seamless way, 2) these models must be provided and integrated in a standardized and auditable way. A generic and easy to customize design, which allows the applicability of the architecture to different use cases must be ensured.
Fast Response Time and Usability: Both of these issues are of fundamental importance when it comes to user acceptance of the provided application. Ideally, response times for requests and searches are nearly real-time and a comfortable user interface is provided.
Accuracy and Result Quality: Models generating results must be highly performant and adjustable, in case of new information becomes available. Therefore, capable resources for retraining and evaluation of models must be in reach, also during active service usage.
Robustness and Restorability: The whole architecture must be robust in terms of workload as well as in terms of restorability. Trained models are versioned, all code is deployed in an automatized way and the execution environment, as well as resources, are persisted to ensure the possibility of recovery from scratch.

In the following, the selected Deep Learning model for computing multi-modal embeddings as well as a blueprint architecture for implementing a retrieval system based on such embeddings is presented.

4 VSE++ for Deep Multi-Modal Embeddings

VSE++[Fa18] is a state-of-the-art and well documented (in the scientific literature, on *GitHub*, and various blog posts) multi-model retrieval model based on *Pairwise Learning* for transforming images and text documents into a joint vector space. Such embeddings can

be efficiently pre-computed, stored, and indexed for later retrieval. Thus, *VSE++* is used as Deep Learning model for the presented architecture.

In detail, *VSE++* transforms an image i with an image encoder $\phi(i, \Theta_\phi)$, whereas a caption c is transformed by a text encoder $\Psi(c, \Theta_\psi)$ to obtain the corresponding vector embeddings. Θ_ϕ and Θ_ψ express the model parameters of the image and the text encoder, respectively. Subsequently, these embeddings are mapped into a joint vector space:

$$f(i, W_f, \Theta_\phi) = W_f^T \phi(i, \theta_\phi), \tag{1}$$

$$g(c, W_f, \Theta_\psi) = W_f^T \psi(i, \theta_\psi). \tag{2}$$

The similarity within the joint vector space $s(i, c)$ can be measured by the *Cosine Similarity*. The model is trained using a loss function focused on hard negatives. Hence, for an image-text pair (i, c) one wants to maximize its dissimilarity with the most similar incorrect image and caption in the batch. The dimensionality of the joint embedding space is set to $n = 1024$. Therefore, a *Resnet152* is used to encode images and a *GRU* without pre-training is used to encode text [He16]. Before feeding images into the encoder, they become centered, randomly cropped, and resized to a shape of $224x224$. In order to preprocess related captions, a vocabulary was determined by using available training data. The text is tokenized using *NLTK* [BKL09], one-hot encoded with respect to our vocabulary, and then passed through an embedding layer to obtain word embeddings of a dimension of 300, which serves as the input of the text encoder.

5 System Architecture of a Deep Learning based Retrieval System

Within this section, an overview of the proposed blueprint architecture for deep, multi-modal information retrieval is given. Figure 1 illustrates its individual components as well as the flow of information between them. In the current use-case, the architecture uses a state-of-the-art Deep Learning model to create multi-modal embeddings. These can be utilized to retrieve text objects corresponding semantically to the given image objects. In general, the architecture consists of two core components: 1) a *Google Cloud* based *Kubernetes* cluster for the orchestration of docker images and the scalable provision of docker containers and 2) an on-premise *GPU Cluster* for the training of Deep Learning models.

5.1 Cloud-based Services in the Kubernetes Cluster

A fundamental corner-stone for orchestrating and deploying a data-product into production is a scalable and flexible container system, which allows rapid up and downscaling for the optimal utilization of resources in an automatized fashion. Currently, *Kubernetes*[3]

[3] https://kubernetes.io

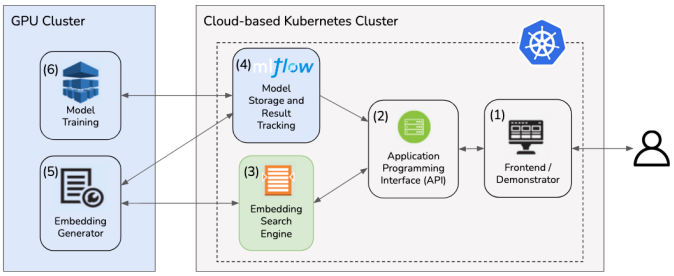

Fig. 1: Overview of the proposed blueprint architecture.

is the de-facto standard for deploying and orchestrating *Docker*[4] containers. A managed *Kubernetes* cluster as Infrastructure-as-a-Service (IaaS) can be easily deployed in all major hyperscaler environments (e.g., *Microsoft Azure*[5], *Amazon Web Services (AWS)*[6], and *Google Cloud Platform (GCP)*[7]). In this work, a managed *Kubernetes* cluster is deployed in the GCP, relying on four small nodes having two virtual CPUs and $8GB$ memory, each. The cluster serves the *1) Frontend/Demonstrator*, the *2) Application Programming Interface (API)*, the *3) Embedding Search-Engine*, and the *4) Model Store and Result Tracking* component. Load balancing is done with *Kubernetes Ingress* by routing external *HTTP(S)* traffic to internal services. Thereby, load balancing enforces a distributed network traffic across multiple workers and it is assured, that a single service never carries an excessive demand. In addition, it offers the flexibility to add or remove nodes and resources based on the current demand.

1) Frontend and Demonstrator with Vue.js In order to allow rapid testing and the acquisition of user feedback concerning the retrieval system's result quality, an adequate frontend is developed and provided to service users.

2) Application Programming Interface (API) To expose the functionalities of the data-product, a backend is implemented. It accepts *REST* requests, encapsulates the computation of embeddings, and queries the *Embedding Search Engine*. It is implemented by using *FastAPI* and operated by using *Uvicorn* — an *Asynchronous Server Gateway Interface* (ASGI) server, which is an asynchronous, performance optimized, and Python based web server. The entire backend is bundled as a *Docker* image and deployed into the *Kubernetes* cluster. There are two exposed interfaces: (i) to compute embeddings for a given image and (ii) to search for similar documents by using the embeddings.

3) Embedding Search-Engine By using *Pairwise Learning* models with joint embedding spaces for images and text, it is possible to pre-compute and store all vector representations of the underlying retrieval database at once. The retrieval step can be therefore formulated as:

[4] https://www.docker.com/
[5] https://portal.azure.com
[6] https://aws.amazon.com
[7] https://cloud.google.com

(i) computing an embedding of an image and (ii) finding the most similar text embedding among all pre-computed embeddings in the collection of relevant documents. To search for most-similar documents using a given embedding, there are several popular solutions: *Faiss* [JDJ19], *ScaNN* [Gu20], (experimental) extensions on ordinary SQL databases such as PostgresSQL[8], and even approaches based on *SQLLite* such as Magnitude [Pa18]. For the architecture presented within this work, *Elasticsearch*[9] was used. Since *Elasticsearch* version 7.3, embeddings can be indexed as vectors and advanced queries on vectors based on the *Cosine Similarity*, the dot product, the *l1*, and the *l2* norm have been introduced. Thereby, pre-computed embeddings from *VSE++* can be stored directly in the search engine. Besides its popularity and adaption in many enterprise environments, *Elasticsearch* scales easily and showed promising performance results within the experiments in Section 6 for indexing and retrieving in a collection of ten thousands of embeddings.

4) Model storage and tracking of results with MLflow Experiments and trained models for computing multi-modals embeddings are stored in the model management service *MLflow*[10]. In particular, the following three features of *MLflow* are of specific importance: *MLflow Tracking*, *MLflow Models*, and the *Model Registry*. Trained models are serialized in the environment-independent *torchscript* format, a unified framework to deploy models for inference in production without requiring the source code or an installed version of *PyTorch*. These models can be easily tagged and used for generating embeddings. *MLflow* comes with a *Python Application Programming Interface*, a user interface, as well as extensions for *PyTorch*, which simplifies the integration into the proposed system.

IT-Engineering and Operations All source code of the above mentioned workflows is versioned in *Gitlab*. Automated CI/CD processes build and test docker images as well as deploy them into *Kubernetes* or the *GPU Cluster*.

5.2 Deep Learning Infrastructure

The model training is done on an on-premise *GPU Cluster*: standard Linux machines (Intel i7-5930K) with $64GB$ RAM containing two *nVidia Titan X* graphics cards with $12GB$ video memory, each. Parallel GPU computation is realized by using *Cuda*[11]. Depending on data and information policies as well as the capabilities of hosting a *GPU Cluster*, the model training can also be migrated to a cloud service as well. Within the *GPU Cluster*, there are two relevant components: the *5) Embedding Generator*, and the *6) Model Training* (see Figure 1).

[8] For example *pgvector*, see `https://github.com/ankane/pgvector`

[9] `https://www.elastic.co`

[10] `https://www.mlflow.org/`

[11] `https://developer.nvidia.com/cuda-zone.`

5) Embedding Generator Here, the latest, stable model from *MLflow* is retrieved and applied to generate embeddings from a given collection of documents. Subsequently, these embeddings are stored within the search engine.

6) Training of Machine Learning Models with PyTorch Workflows for training and evaluating *VSE++* model are based on *PyTorch Lightning5*. For model training itself, workflows adapted by the *VSE++* authors are used. Newly trained models are evaluated based on the retrieval metrics *Precision@ k* and *Recall@k* (P@K, and R@k).

6 Results and Discussion

Datasets	Size (images)	Train	Validation	Test
Flickr30k	31.783	29.783	1.000	1.000
COCO	123.287	113.287	5.000	5.000
Total	**155.070**	**143.070**	**6.000**	**6.000**

Tab. 1: Dimensions and splitting of the combined dataset for model training and evaluation, consisting of *MS COCO* and *Flickr30k*.

In the following, the features of the proposed blueprint architecture introduced in Section 5 are reviewed, discussed, and matched in terms of fulfilling the requirements stated in Section 3.

To evaluate the performance we trained *VSE++* and applied the proposed system onto the *Microsoft Common Objects in Context* (MS COCO) and the *Flickr30k* datasets, which can be regarded as the state-of-the-art datasets for performance measurement within this domain. They are consisting of 155.070 matched images and text documents (see Table 1).

Generalizability The application is decomposed into a set of manageable services which are faster to develop, easier to understand, and can be updated independently. Thereby, the underlying micro-service architecture allows an easy transformation and adaption to a broad variety of different applications and services. Moreover, the modularity of containers orchestrated by *Kubernetes* within the *GCP* enables a fast exchange and up scaling of all functionalities.

Affordability Table 2 gives a brief overview of the average monthly costs for the proposed architecture. These costs provide a broad overview, while exact pricing may change due to selected hyperscaler, used computing engines, storage accounts, etc. They were accrued from running all the services of the entire application for one month without auto-scaling. The major costs are associated with components delivered by the *GCP Compute engine*. Additionally, expenses for the on-premise *GPU Cluster* must be considered, which would be added to the expenses of the cloud platform if hosted there. In a production environment, the utilization of features like *Kubernetes* auto-scaling allows the available resources to both, grow and shrink, which leads to correlating costs into both directions. Thereby, the

Service	Resource Type	Subtotal
Compute Engine	E2 Instance Core	153.36€
Compute Engine	E2 Instance Ram	82.20€
Compute Engine	Storage PD Capacity	26.47€
Compute Engine	Network Load Balancing	17.82€
Compute Engine	Misc. Networking Services	14.58€
Cloud Storage	Download Worldwide Destinations	18.88€
Cloud Storage	Standard Storage	0.71€
Cloud Storage	Regional Standard Class B Operations	0.09€
	Total	**314.11€**

Tab. 2: A high-level overview of the average operational costs for one month.

expenses scale with the demand to the platform, which means if a business is small or a new business idea is formed, it can be brought into production in an affordable way. If the demand gets higher due to heavy usage of the service, it may become more expensive, but also the revenue created by the provided service may be larger.

Scalability By leveraging the *Kubernetes* capabilities, the blueprint benefits from technology agnostic frameworks. Thereby, the flexibility to deploy any application resource or web framework needed for the provision of a service is given. Due to features like service discovery, load balancing, horizontal scaling, and self-healing, the utilization of *Kubernetes* is another important factor for achieved scalability, high-availability, and fault tolerance.

Modularity and Extensibility Developing Deep Learning based applications and bringing them into production raises several new challenges. The Machine Learning lifecycle, a complex and iterative process including data collection, model training, verification, deployment, and monitoring must be considered [ACP19]. To ensure a fluent model deployment, reproducibility, and an easy expansion of the application, standardized model versioning and model management is important. To overcome these challenges, *MLflow Models* and *Model Registry*, both part of *MLflow*, are used. Thereby, trained models can be packaged, versioned, and utilized in an easy fashion. To deploy them in a unified way, they become serialized with *torchscript*. Thereby, the serialized model is self-contained including all requirements and the process itself is generic. Due to the versioning within the Model Registry, the deployment of most recent models, as well as a rollback to previously deployed versions, are feasible. Moreover, tagging and seamless deployment of models into the system without any additional effort is possible.

Fast Response Time and Usability In order to make a user accept a service, it must be able to offer high quality results and to deliver results with a rapid response time. In the context of this work, the fast retrieval of the textual results is based on the similarity of joint vector embeddings that offer the advantage of pre-computation. Due to only processing incoming request documents, the response time is lowered significantly. In order to store and search for pre-computed embeddings, *Elasticsearch* which is capable of storing and retrieving

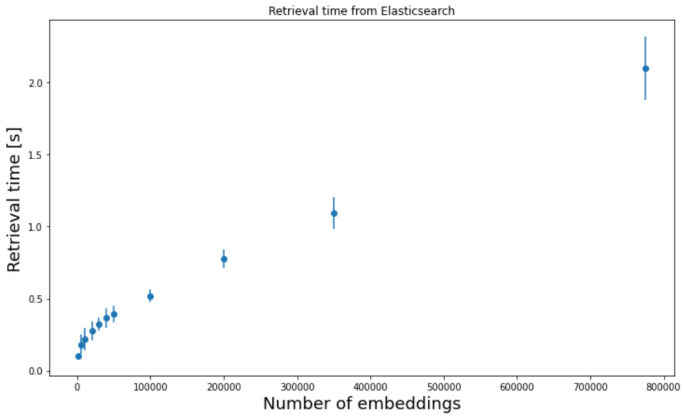

Fig. 2: Resulting retrieval time of *Elasticsearch* as an embedding search engine containing a varying number of pre-computed multi-modal embeddings.

dense vectors in a distributed way is used. Figure 2 shows the result retrieval times for a different number of pre-computed and stored vector embeddings. Within a range $0 - 10.000$ embeddings, the retrieval time takes significantly less than $1s$. For a number of more than 70.000 embeddings, the retrieval time ranges around 20 seconds, which is still comparably low concerning the huge amount of stored information. These results prove the adequate capabilities of the embedding search engine.

Additionally, the integration of *Elasticsearch* into *Kubernetes* is comparably easy, it is well-established, matured, and offers a decent documentation as well as support. Next to a fast response time and a decent result quality, the provision of an intuitive, appealing, as well as functional user interface is mandatory for a high usability and the acceptance of a service by potential users.

Model	**R@1**	**R@3**	**R@5**	**P@1**	**P@3**
VSE++	0.343	0.542	0.634	0.343	0.388

Tab. 3: Results for training *VSE++* on a combined dataset consisting of *COCO* and *Flickr30k*, see Figure 3.

Accuracy and Result Quality In order to deliver high quality results, models can continuously be trained on new data, even during times of active service operation. This is ensured by providing a separate computation instance in terms of the *GPU Cluster*. With this feature in combination with the options of versioning and delivering new models via *MLflow*, an optimal result quality can be guaranteed.

Table 3 shows the results of the *VSE++* model for the retrieval metrics *Precision@k* for *k=1* and *k=3* as well as *Recall@k* with *k=1*, *k=3*, and *k=5*. The achieved results of the re-implemented models used in the blueprint architecture resemble with the ones achieved by authors of *VSE++* [Fa18].

Robustness and Restorability In terms of robustness, the architecture is backed by the *Kubernetes* auto scaling and load balancing features. Due to the appliance of state-of-the-art *DevOps* tools like *Gitlab*, and *Gitlab CI* the complete restorability of the needed infrastructure is ensured. In order to also preserve the underlying resource architecture, tools like *Terraform* could be applied. Moreover, the usage of unit and integration tests during the deployment process assures the correct functionality of the published code.

7 Conclusion and Outlook

We presented a scalable, modular, and highly responsive multi-modal retrieval system — a retrieval system for seamless information retrieval from different domains such as text, image, and audio — based on deep learned embeddings and loosely coupled services in a *Kubernetes* cluster. Despite the *Kubernetes* cluster was deployed in a *IaaS* manner into the *GCP*, its design is generic and can be deployed in any other cloud or on-premise environment. GPU intensive Deep Learning training and indexing were conducted on dedicated hardware; this can also be done in the cloud, either directly on the *Kubernetes* cluster with additional GPU resources, or via services such as Google Colab[12]. The multi-modal retrieval step is solved by using the VSE++ Deep Learning model, which maps documents from different domains into a common vector space. Thereby, semantically similar features are mapped to similar locations. The *Pairwise Learning* approach VSE++ provides reasonable good retrieval results and requires comparable few computational power for computing embeddings. To evaluate our implementations and benchmark the system the *COCO* and the *Flickr30k* dataset (consisting of matched images and text documents) were used. Finally, with *Elasticsearch*, *MLflow*, and an implemented REST-based backend the draft for a general, scalable, and cost-efficient architecture for a multi-modal retrieval system was realized. The presented architecture can be utilized and extended for any multi-modal information retrieval system. Thus, this blueprint can be used as a generic guideline by small as well as medium-sized companies to bring their Deep Learning applications into production.

Acknowledgement The findings and concepts presented in this work are partially funded by the *Service-Meister* project - funded by the *Innovationswettbewerb KI* and the *Bundesministerium für Wirtschaft und Energie* of the Federal Republic of Germany.

Bibliography

[ACP19] Ashmore, Rob; Calinescu, Radu; Paterson, Colin: Assuring the machine learning lifecycle: Desiderata, methods, and challenges. arXiv preprint arXiv:1905.04223, 2019.

[BKL09] Bird, Steven; Klein, Ewan; Loper, Edward: Natural language processing with Python: analyzing text with the natural language toolkit. Ö'Reilly Media, Inc.", 2009.

[12] `https://colab.research.google.com/`

[Ch20] Chen, Jianan; Zhang, Lu; Bai, Cong; Kpalma, Kidiyo: Review of Recent Deep Learning Based Methods for Image-Text Retrieval. In: 2020 IEEE Conference on Multimedia Information Processing and Retrieval (MIPR). IEEE, pp. 167–172, 2020.

[Fa18] Faghri, Fartash; Fleet, David J; Kiros, Jamie Ryan; Fidler, Sanja: VSE++: Improving Visual-Semantic Embeddings with Hard Negatives. In: Proceedings of the British Machine Vision Conference (BMVC). 2018.

[Gu20] Guo, Ruiqi; Sun, Philip; Lindgren, Erik; Geng, Quan; Simcha, David; Chern, Felix; Kumar, Sanjiv: Accelerating Large-Scale Inference with Anisotropic Vector Quantization. In: International Conference on Machine Learning. 2020.

[He16] He, Kaiming; Zhang, Xiangyu; Ren, Shaoqing; Sun, Jian: Identity mappings in deep residual networks. In: European conference on computer vision. Springer, pp. 630–645, 2016.

[HW19] Huang, Yan; Wang, Liang: Acmm: Aligned cross-modal memory for few-shot image and sentence matching. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 5774–5783, 2019.

[JDJ19] Johnson, Jeff; Douze, Matthijs; Jégou, Hervé: Billion-scale similarity search with GPUs. IEEE Transactions on Big Data, pp. 1–1, 2019.

[Li19] Li, Kunpeng; Zhang, Yulun; Li, Kai; Li, Yuanyuan; Fu, Yun: Visual semantic reasoning for image-text matching. In: ICCV. 2019.

[Li20] Li, Xiujun; Yin, Xi; Li, Chunyuan; Zhang, Pengchuan; Hu, Xiaowei; Zhang, Lei; Wang, Lijuan; Hu, Houdong; Dong, Li; Wei, Furu et al.: Oscar: Object-Semantics Aligned Pre-training for vision-language Tasks. In: European Conference on Computer Vision. Springer, pp. 121–137, 2020.

[Pa18] Patel, Ajay; Sands, Alexander; Callison-Burch, Chris; Apidianaki, Marianna: Magnitude: A Fast, Efficient Universal Vector Embedding Utility Package. In: Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations. Association for Computational Linguistics, Brussels, Belgium, pp. 120–126, November 2018.

[Sc15] Sculley, David; Holt, Gary; Golovin, Daniel; Davydov, Eugene; Phillips, Todd; Ebner, Dietmar; Chaudhary, Vinay; Young, Michael; Crespo, Jean-Francois; Dennison, Dan: Hidden technical debt in machine learning systems. Advances in neural information processing systems, 28:2503–2511, 2015.

[Wa19] Wang, Zihao; Liu, Xihui; Li, Hongsheng; Sheng, Lu; Yan, Junjie; Wang, Xiaogang; Shao, Jing: Camp: Cross-modal adaptive message passing for text-image retrieval. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 5764–5773, 2019.

[ZL18] Zhang, Ying; Lu, Huchuan: Deep cross-modal projection learning for image-text matching. In: Proceedings of the European Conference on Computer Vision (ECCV). pp. 686–701, 2018.