

Contents

1 Introduction	3
1.1 Background	3
1.2 Motivation	3
1.2.1 Need for Enhanced Interactivity in 3D Environments	3
1.2.2 Potential of Large Language Models	3
1.3 Research Objectives	3
1.4 Thesis Structure	3
2 State of the Art and Related Work	4
2.1 Large Language Models	5
2.1.1 Evolution and Capabilities	5
2.1.2 Transformer and Self-Attention	6
2.1.2.1 Seq2Seq without Attention	6
2.1.2.2 Basic Attention	7
2.2 Attention is All You Need	8
2.2.1 Application	10
2.2.1.1 Embedding	10
2.2.2 Usage Techniques	10
2.3 Interactive 3D Technology	10
2.3.1 Fundamentals of 3D Technology	10
2.4 Integration of LLMs in 3D Scenes	11
2.5 3D Model File Formats	11
2.5.1 glTF (GL Transmission Format)	11
2.5.2 Overview of File Format Types	11
2.5.3 Geometry and Texture Data	11
2.5.4 Animation and Rigging Support	11
2.6 Labeling 3D Models	11
2.7 Related Work	11
2.7.1 Cap3D	11
3 Concept and Architecture	13
3.1 System Architecture	13
3.2 Interaction Design	13
3.3 Understanding Object Relationships	13
3.4 User Interface	13
3.5 LLM Querying Mechanism	13
4 Realization	14

4.1 Labeling and Enriching 3D Models	14
4.1.1 Labeling Process	14
4.1.2 Enrichment Techniques	14
4.2 Efficient LLM Querying	14
4.3 LLM Integration	14
4.4 User Interface and Interaction	14
4.5 Technical Challenges and Solutions	14
5 Evaluation	15
5.1 Methodology	15
5.2 Results	15
5.3 Discussion	15
6 Conclusion and Outlook	16
6.1 Labeling	17
6.1.1 Methodology	18
6.2 Vector Database	20
Bibliography	i

1 Introduction

1.1 Background

1.2 Motivation

1.2.1 Need for Enhanced Interactivity in 3D Environments

1.2.2 Potential of Large Language Models

1.3 Research Objectives

1.4 Thesis Structure

2 State of the Art and Related Work

2.1 Large Language Models

2.1.1 Evolution and Capabilities

Language models are computer-based systems designed to comprehend and produce human language. These models can either predict the probability of word sequences or create new text from provided input. Despite their transformative potential, early language models encountered several obstacles including handling uncommon or novel words, avoiding overfitting, and grasping intricate linguistic details. [1]

The evolution of language models has been marked by several significant stages of development. Initially, the focus was on statistical language models, followed by neural language models which leveraged deep learning techniques. The introduction of pre-trained language models marked a pivotal shift, particularly with the development of BERT. BERT utilized the Transformer architecture, which is highly parallelizable and includes self-attention mechanisms. This model was groundbreaking in setting the “pre-training and fine-tuning” paradigm, which became a foundational approach for subsequent models like BART and GPT-2 . [2]

Large language models (LLMs) represent a further advancement in the field, characterized by their vast parameter sizes and enhanced learning capacities. Researchers discovered that increasing the scale of these models, whether in model size or data size, generally improves their performance on various tasks. These larger models, such as the 1.5 billion-parameter GPT-2, exhibit unique behaviors and emergent abilities that are not evident in their smaller counterparts. This scaling up has led to the coining of the term “large language models” or LLMs. [2]

A key attribute of LLMs is their capability for in-context learning, where the model generates text based on a specified context or prompt. This feature is crucial for producing coherent and contextually relevant responses, making LLMs well-suited for interactive and conversational applications. For example, models like GPT-3 and PaLM are trained to predict the next token in a sequence, maximizing the probability of this token given the context. This training exploits the Transformer architecture’s self-attention module, which is vital for handling sequential data efficiently and capturing long-range dependencies in text. [1]

2.1.2 Transformer and Self-Attention

- (todo: we will refer to sequence as a sentence in the following)
- (todo: further we abstract some details like the fact that we dont use words as input but instead word embeddings)

Transformers are vital to the success of LLMs. One major advantage is that they allow for massive parallel computation capabilities and therefore use of GPU hardware. Another advantage of transformers is that they can look at the input at once at each timestep [correct this/make it more accurate]. The Transformers (architecture?) was introduced in the paper “Attention is all you need”. In order to explain why we need attention it is important to understand what problem it solves. To do that we will look at a simple model that takes a sequence of text and outputs a sequence of text, which can be used for example for machine translation, a so called seq2seq encoder-decode architecture. Which as the name indicates first encodes the sequence into a different representation using the encoder and then decodes the numeric representation back into a sequence.

2.1.2.1 Seq2Seq without Attention

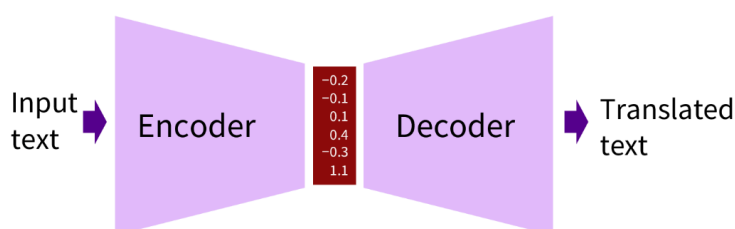


Figure 1: simplified seq2seq encoder-decoder architecture

In order to understand how the encoder-decoder architecture works we will look at how one could use it for machine translation with the goal to give for example an English text sequence as input and receive a German output. During training we would teach the encoder-decoder-model to output the same text as it gets as input. This way we force the model to learn to encode properly, that is to encapsulate all the meaning of a sequence into a numerical representation. The decoder also learns how to decode this representation back to the original sentence.

If we want to build an English-German translator we could train an encoder-decoder on the English training data and a separate pair of encoder-decoder on the German language. Since we know that the English encoder can take the text into a context vector that contains the meaning and we know that the German decoder can decode a context vector and its

meaning back into a sentence We can pair the english encoder with the german decoder to receive a english-german translator.

2.1.2.2 Basic Attention

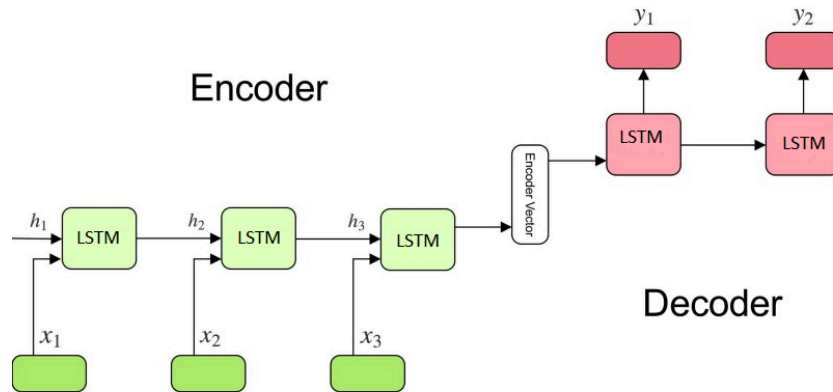


Figure 2: more detailed encoder decoder

However this approach has a multiple flaws. A major one is the context vector as it becomes the bottleneck. If we look more closely how the internals of the encoder decoder look like in figure 2. We can see that the encoder at each timestep takes the next word and the context up to that point which from the previous node and then itself passes the new context to the next node. (We refer to nodes and context nodes which are on finer level in this case a LSTM node its the hidden layer.) The issue with this approach since the context is updated during the processing the meaning at the beginning of sequence is often lost.

To resolve this issue attention was implemented in (paper). To combat the context vector bottleneck and the forgettness of the model we could introduce attention during the decode step as illustrated in [].

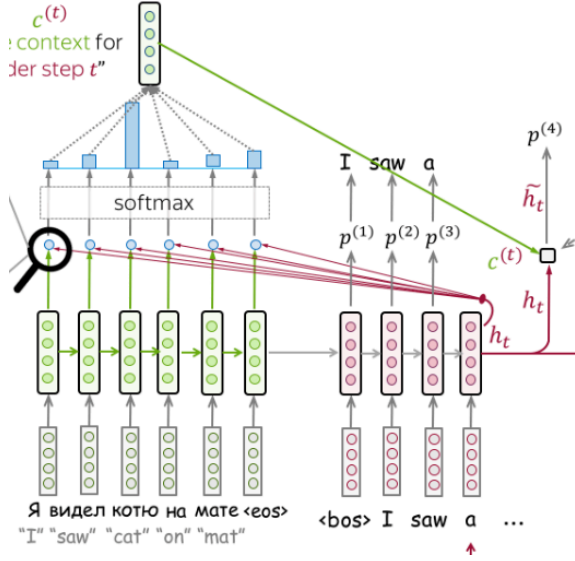


Figure 3: attention for encoder-decoder interaction

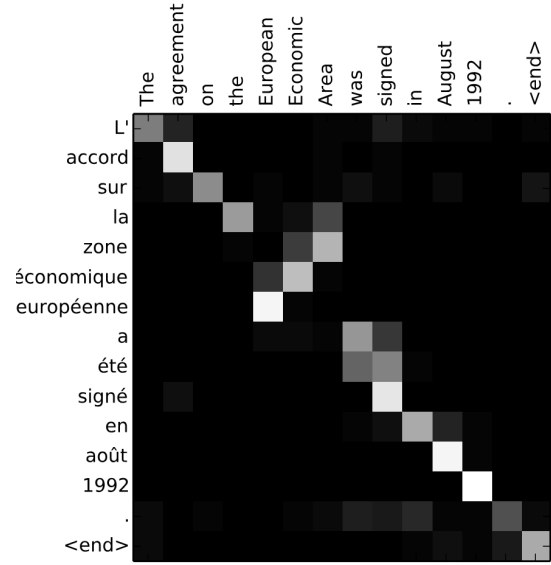


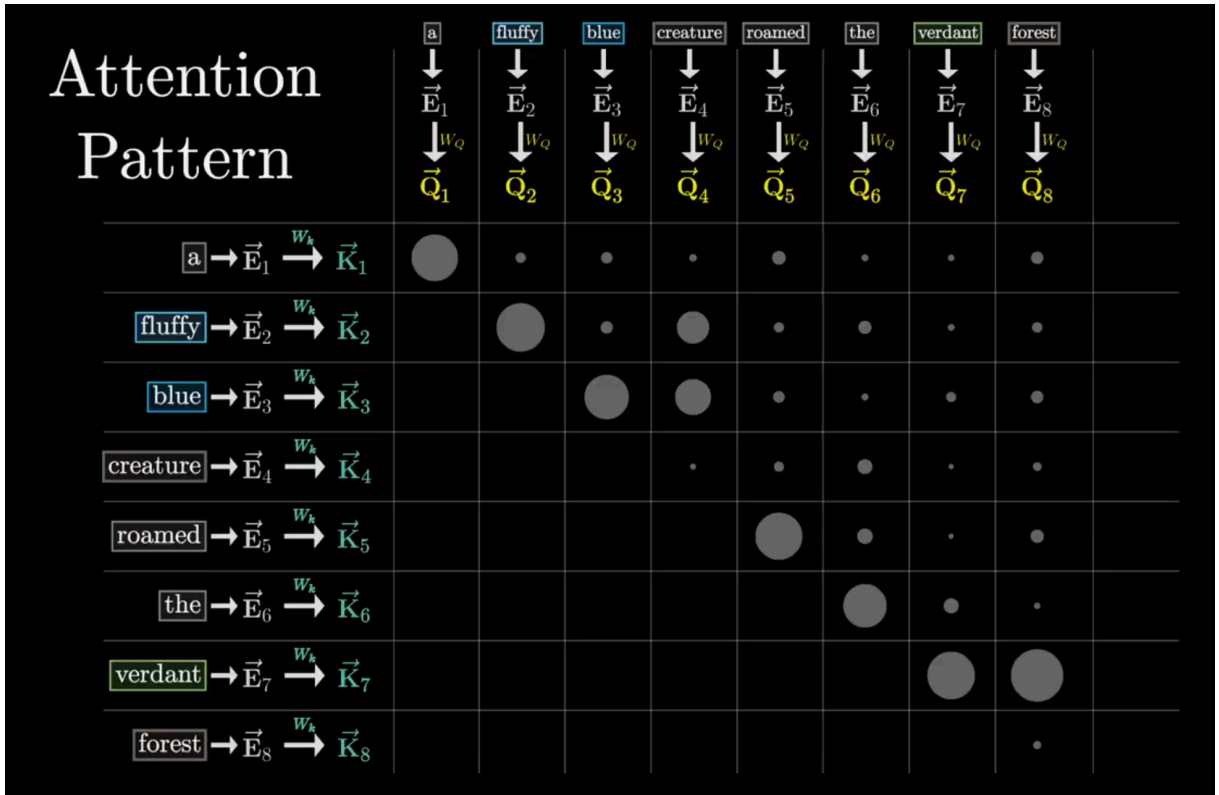
Figure 4: attention-matrix heatmap

We can do this by looking at a decode timestep which input elements/context are relevant to generate the next word. During each timestep where we decode a word. We compare the context vector we would typically receive against all the hidden states generated by encoder and apply a score function. A simple to understand score function for our purposes would be the dot product. The dot product returns a numerical value which represents how closely two vectors align. When we then apply softmax we get for a timestep how relevant as a percentage each input is for the generation of the next timestep. This is referred to as cross-attention since the decoder “pays attention” to the encoder. A visualization of attention can be seen in figure 3. Here we can see a translation from French to English. We can see that the model pays attention for the word “la” when generating “bla”.

2.2 Attention is All You Need

The paper in which Transformers are introduced is called (insert here) For our example we used a Recurrent Neural Network. A Recurrent Neural Network is a network where for each timestep we use the result of the previous timestep. Our previous seq2seq model with attention has a RNN encoder, a RNN decoder and attention as the decoder-encoder interaction mechanism []. As the name of the paper suggests the transformer however uses attention for encoding, decoding and interaction between them. The following outlines how this works. An important aspect is self-attention instead of ... we use attention inside the encoder itself. instead of using an RNN we use a model where elements are not processed one after the other but instead in parallel. as depicted in

figure 6 simplified we can see that the first layer takes in all words at once. then each input gathers context from all the other tokens resulting in a updated representation in a layer above which should not only encapsulate the meaning of the word but also the meaning of the word in context with the rest of the input. for example the word “bank” could change its meaning depending on the context from furniture to a financial institution. Similarly one could imagine that adjectives might be incorporated in the meaning on the next higher layer of a word. figure 6 abstracts how attention is calculated for simplisty. To calculate attention the transformer has additional model weights and is calculated as follows: $\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{Q \cdot K^T}{\sqrt{d_k}}\right)$ With Q, K, V standing for Query, Key and Value weight matrices. Since these weights are calculated during training it is not possible to tell how they work exactly but we can gain a create a simplified example on how these matrixes might work. We will use a example from [1]. Lets say we have the sentence “a fluffly blue creature”, we want the first layer after the self attention to encode the fact that the creature is blue and fluffly. when we look at the forth word “creature” the embedding in the second layer should also encode blue and fluffly. The query could be “



Using this Key, Value approach we figure out which words relate to which other words.

asd

2.2.1 Application

-  **RAG**
-  **Vector DB**

Tokenization

Tokenization in natural language processing refers to the process of breaking down text into smaller units called tokens for subsequent analysis and processing. This step is crucial for various NLP tasks like text understanding, information extraction, and language modeling [3]. Different algorithms, including data-driven subword-based approaches like Byte Pair Encoding (BPE) and expert-crafted segmentation methods, are used for tokenization in handling diverse languages and text structures [4]. Tokenization is a fundamental component in NLP, enabling effective text manipulation and enhancing the performance of language models such as BERT and GPT [5], [6].

-  **introduce bpe and why important**

Byte Pair Encoding (BPE) is a method that constructs a vocabulary by merging the most frequent characters or character sequences to create new tokens iteratively. It involves an iterative process of removing tokens based on a formula incorporating token frequency and a hyperparameter, α . Sennrich et al. introduced this technique in 2016, using byte pairs to efficiently tokenize text [7]. The approach enhances multilingual language processing by manipulating text similar to image and speech processing techniques [8].

2.2.1.1 Embedding

-  **Embedding**

2.2.2 Usage Techniques

2.3 Interactive 3D Technology

2.3.1 Fundamentals of 3D Technology

-  **difference: Multi-view Images, Volumetric, Point Cloud, Polygonal Mesh, Primitive-based**
-  **meshes, textures**
-  **animation, rigging**
-  **inverse kinematics**
-  **raycasting/linecasting**

2.4 Integration of LLMs in 3D Scenes

2.5 3D Model File Formats

2.5.1 glTF (GL Transmission Format)

The GL Transmission Format (glTF) is a file format designed for the efficient transmission and loading of 3D scenes and models by applications. It is a JSON-formatted file for describing the scene, which can include buffer data, shaders, and textures stored in separate binary files, aiming for rapid previewing of 3D geometry and shading [9]. Introduced by the Khronos Group in October 2015, glTF has been developed to be web-friendly and supports extensions like binary glTF (GLB) for merging all components into a single binary blob, enhancing its efficiency for web-based delivery and fast rendering [10], [9]. Despite its capabilities, glTF is not considered a complete scene-graph as it lacks built-in support for certain features like lights, necessitating extensions [9].

2.5.2 Overview of File Format Types

-  **table**



2.5.3 Geometry and Texture Data

2.5.4 Animation and Rigging Support

2.6 Labeling 3D Models

2.7 Related Work

2.7.1 Cap3D

-  **add citation for the models**
-  **image of stages**

The paper “Scalable 3D Captioning with Pretrained Models” presents Cap3D, a methodology designed to generate captions for 3D objects utilizing pretrained models from different domains such as image captioning, image-text alignment, and large language models (LLMs). This method aims to automate the process of caption generation by integrating captions from multiple viewpoints of a 3D asset, circumventing the need for manual annotation.

-  **image of perspectives**

The Cap3D approach is structured in several stages. Initially, multiple 2D views of each 3D object are rendered using Blender to capture details from varied perspectives. These

images are then processed through the BLIP2 image captioning model to produce multiple candidate captions for each view. Due to potential inaccuracies in the generated captions, the CLIP model is used in the subsequent step to evaluate and select the most accurate caption based on its semantic similarity to the image encoding.

The final step in the methodology involves consolidating the selected captions into a cohesive and comprehensive description using the GPT4 model. This step synthesizes the information across different views into a single caption that accurately reflects the 3D object’s characteristics.



The application of Cap3D was primarily demonstrated on the Objaverse dataset, leading to the creation of a large-scale dataset comprising 660,000 3D-text pairs. This dataset was further subjected to ethical filtering to ensure the appropriateness of the captions. The dataset, along with associated tools and models, has been made publicly available to facilitate further research.

Cap3D was evaluated using a subset of the Objaverse dataset containing 41,000 human annotations, assessing its quality, efficiency, and cost relative to human-generated captions. Additionally, its ability to describe complex geometries was tested on the ABO dataset. The study also noted that text-to-3D models finetuned with Cap3D captions outperformed those trained with human captions, suggesting the method’s potential to enhance 3D modeling. Overall, Cap3D automates 3D object captioning by effectively utilizing various AI models, thereby improving the annotation process’s scale and efficiency.

3 Concept and Architecture

3.1 System Architecture

3.2 Interaction Design

-  **mockup**
-  **tts, speech2text**

3.3 Understanding Object Relationships

3.4 User Interface

3.5 LLM Querying Mechanism

4 Realization

4.1 Labeling and Enriching 3D Models

4.1.1 Labeling Process

4.1.2 Enrichment Techniques

4.2 Efficient LLM Querying

4.3 LLM Integration

4.4 User Interface and Interaction

4.5 Technical Challenges and Solutions

5 Evaluation

5.1 Methodology



5.2 Results

5.3 Discussion

6 Conclusion and Outlook

6.1 Labeling



The primary limitation of large language models (LLMs) is their inherent reliance on textual data. This constraint presents a significant challenge when attempting to apply LLMs to non-textual domains such as 3D scenes. To effectively integrate LLMs into 3D scene analysis and interaction, it is imperative to translate the complex, spatially oriented data of a 3D scene into a structured text format that an LLM can process.

-  maybe add image
-  maybe what is Sketchfab

This necessitates the use of 3D assets that are thoroughly labeled and annotated. However, to access a wide variety of 3D assets, platforms like Sketchfab are highly advantageous, as they feature user-shared 3D assets often under permissive licenses. Despite this accessibility, it is important to note that models from such platforms frequently lack comprehensive and precise semantic labels and may require further refinement to be fully utilizable by LLMs.

-  touch up the paragraph bellow

While 3D assets from community platforms generally have titles, their individual components or nodes often lack proper labeling. The task of labeling and captioning is a domain where the use of machine learning (ML) models has become the standard approach due to their automatic feature recognition capabilities. In the realm of computer vision, a variety of models are available that can be effectively applied for different tasks such as object detection, image-to-text conversion, and more. Here's a brief overview of some key types of computer vision models, categorized by their typical inputs and outputs:

-  improve, add, make smaller
-  add caption

Model Type	Input	Output
Image Classification	Single image	Categories (labels for what is depicted)
Object Detection	Single image	Bounding boxes with labels per object
Image Segmentation	Single image	Pixel-wise masks categorizing each pixel
Image-to-Text (Captioning)	Single image	Descriptive text of the image
Point Cloud Segmentation	3D point cloud data	Categorized point clusters
RGB-D Object Detection	RGB images + depth data	Bounding boxes with depth information

-  this does not belong here -> practical part

Selecting the appropriate model type for a project involves several key considerations, such as compatibility with the data type, model accuracy, resource availability, and the specific task at hand. A major consideration for this thesis was computing efficiency,

primarily due to the need for models that can operate within the constraints of available computational resources, like GPU memory.



For this project, models that require inputs other than images were deemed less suitable due to limitations in VRAM capacity and the scarcity of pre-trained models capable of efficiently processing complex data types like point clouds or RGB-D images. Furthermore, when experimenting with segmentation models, a significant challenge was mapping the segmented outputs back onto the 3D asset and its individual nodes accurately.

Initial trials with classification models revealed issues with their limited vocabulary and a tendency towards more generic descriptors. Consequently, an image captioning model was chosen for its ability to generate more descriptive and contextually relevant captions.

6.1.1 Methodology

Write something along the lines:

- we wanted to have a solution where given a scene e.b. a classroom all objects are labeled
- as well as label individual parts i.e. nodes of an object for example given an engine we want to automatically label the pistons
- cap3d is only able to label individual object
- since our approach given a 3d file not only has to generate one label but one label for potentially all the nodes in that file -> we decided to develop our own lightweight variant of cap3d
- write about how cap3d uses blender and we did not want to use blender as it is heavy and could potentially hinder multithreading
- our approach is inspired by cap3d
- it consists of three steps: rendering, captioning, filtering
- the rendering is done using pyrender
- using the python libraries trimesh and pyrender we load a scene
- we then set up the lighting by using directional lights using spherical coordinates with rotation matrices so that point into the center of the hypothetical sphere
- Iterate through the nodes of the scene to identify mesh nodes
- And for each node we prepare a list of tasks to take advantage of hyperthreading using multiprocessing with a pool of worker processes
- For each mesh, compute multiple camera positions


- For each camera position, set the camera's pose and render the scene to an image file.
- The images are saved with filenames indicating the mesh index and camera position index.
- for image captioning we use the efficientnet model from google which strikes a good balance of efficiency and performance [citation needed].
-  **how to continue... it was mostly a failure**
-  **cite models, reference cap3d chapter**

d

6.2 Vector Database

Traditional database management systems are primarily designed for structured data and often struggle to effectively handle unstructured data, such as images, audio, and text, which require more complex processing to extract meaningful information.





Write something alike:

- Vector databases on the hand are typically used for use with unstructured data
- Vector databases are good at finding similarites with unstructured data
- They enable use cases like image, video search and recommender system as well as fraud and anomaly detection []
- Vector databases function by first generating vectors by applying some kind of transformation or embedding function to the raw data [11]
- Vector embeddings transfer discrete data into n-dimensionsal vector space called the latent space. [12]
- Typically we want our vectors to be dense representations of the original data in order to allow for efficient comparisons and querying
- The goal is to construct vector embeddings where the geometry in the latent space reflects semantic relationships
-  **add image here**
- Vector embeddings place related items close together in a space, using distances and directions to represent different types of relationships between them. For example, in vector embeddings, synonyms like “happy” and “joyful” may be close together and we may expect the vector $v := f(\text{Paris}) - f(\text{France})$ to be approximately equal to $f(\text{Berlin}) - f(\text{Germany})$ [12].
- Depending on the task and data type there are different ways to achieve a vector embedding but typically deep neural networks are used and their capability to extract semantic features.

todo

- For example to embedd images into vector space one can use the weights during inference of a pre-trained CNN (Convolutional Neural Network) model up to the last layer before classification, also called the penultimate layer. Since the first layers of a CNN (Convolutional Neural Network) extract the features of the input image. [] Before the subsequent layers progressively refine these features into higher-level

representations before classification. Thus we have a feature rich yet dense vector representation of our image.

-  **which word embeddings are used**
-  **knn and ann**
-  **faiss and hns**
-  **rag with image**

Bibliography

- [1] Y. Chang *et al.*, “A Survey on Evaluation of Large Language Models,” *ACM Transactions on Intelligent Systems and Technology*, vol. 15, no. 3, pp. 1–45, 2024, doi: [10.1145/3641289](https://doi.org/10.1145/3641289).
- [2] W. X. Zhao *et al.*, “A Survey of Large Language Models.” Accessed: Apr. 08, 2024. [Online]. Available: <http://arxiv.org/abs/2303.18223>
- [3] *Natural Language Processing for the Semantic Web*. Accessed: Apr. 17, 2024. [Online]. Available: <https://link.springer.com/book/10.1007/978-3-031-79474-2>
- [4] M. Van Nguyen, V. D. Lai, A. P. B. Veyseh, and T. H. Nguyen, “Trankit: A Light-Weight Transformer-based Toolkit for Multilingual Natural Language Processing.” Accessed: Apr. 17, 2024. [Online]. Available: <http://arxiv.org/abs/2101.03289>
- [5] Y. Tay *et al.*, “Charformer: Fast Character Transformers via Gradient-based Sub-word Tokenization.” Accessed: Apr. 17, 2024. [Online]. Available: <http://arxiv.org/abs/2106.12672>
- [6] J. Singh, B. McCann, R. Socher, and C. Xiong, “BERT is Not an Interlingua and the Bias of Tokenization,” in *Proceedings of the 2nd Workshop on Deep Learning Approaches for Low-Resource NLP (DeepLo 2019)*, C. Cherry, G. Durrett, G. Foster, R. Haffari, S. Khadivi, N. Peng, X. Ren, and S. Swayamdipta, Eds., Hong Kong, China: Association for Computational Linguistics, Nov. 2019, pp. 47–55. doi: [10.18653/v1/D19-6106](https://doi.org/10.18653/v1/D19-6106).
- [7] K. Bostrom and G. Durrett, “Byte Pair Encoding is Suboptimal for Language Model Pretraining.” Accessed: Apr. 17, 2024. [Online]. Available: <http://arxiv.org/abs/2004.03720>
- [8] D. Gillick, C. Brunk, O. Vinyals, and A. Subramanya, “Multilingual Language Processing From Bytes.” Accessed: Apr. 17, 2024. [Online]. Available: <http://arxiv.org/abs/1512.00103>
- [9] J. S. Dhanjan and A. Steed, “Revisiting the Scene-Graph-as-Bus Concept: Inter-networking Heterogeneous Applications Using glTF Fragments,” in *2021 IEEE Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW)*, Lisbon, Portugal: IEEE, Mar. 2021, pp. 342–346. doi: [10.1109/VRW52623.2021.00068](https://doi.org/10.1109/VRW52623.2021.00068).
- [10] T. Scully, S. Friston, C. Fan, J. Doboš, and A. Steed, “glTF streaming from 3D repo to X3DOM,” in *Proceedings of the 21st International Conference on Web3D Technology*, Anaheim California: ACM, Jul. 2016, pp. 7–15. doi: [10.1145/2945292.2945297](https://doi.org/10.1145/2945292.2945297).
- [11] Y. Han, C. Liu, and P. Wang, “A Comprehensive Survey on Vector Database: Storage and Retrieval Technique, Challenge.” Accessed: Apr. 24, 2024. [Online]. Available: <http://arxiv.org/abs/2310.11703>

- [12] M. Grohe, “word2vec, node2vec, graph2vec, X2vec: Towards a Theory of Vector Embeddings of Structured Data,” in *Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, Portland OR USA: ACM, Jun. 2020, pp. 1–16. doi: [10.1145/3375395.3387641](https://doi.org/10.1145/3375395.3387641).