VIETNAM NATIONAL UNIVERSITY HO CHI MINH CITY
HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

REPORT

SPECIALIZED PROJECT

SEMESTER 232 ACADEMIC YEAR 2023-2024

DEVELOP A MULTIMODAL CHATBOT FOR A FASHION STORE

MAJOR: COMPUTER SCIENCE

COUNCIL: COMPUTER SCIENCE - 01 CLC

SUPERVISOR(s): QUAN THANH THO, Ph.D.

SECRETARY: TRAN HUY, MEng.

——o0o——

STUDENT 1: VO HOANG NHAT KHANG - 2152646

STUDENT 2: NGUYEN PHAN TRI DUC - 2152528

HO CHI MINH CITY, May 2024

Instructor's Signature

_____ Date: _____

Assoc. Prof. Quan Thanh Tho, Ph.D. (Instructor)

Associate Professor

Faculty of Computer Science and Engineering

Declaration Of Authenticity

We declare that we solely conducted this specialized project, under the supervision of Assoc. Prof. Quan Thanh Tho at the Faculty of Computer Science and Engineering, Vietnam National University - Ho Chi Minh City University Of Technology.

We have taken care to properly acknowledge and document all external sources and references used in the project.

If there is any instance of plagiarism, we are ready to accept the consequences. Ho Chi Minh City University of Technology - Vietnam National University HCMC will not be held responsible for any copyright violations that may have occurred during my research.

Ho Chi Minh City, May 2024

Authors,

Vo Hoang Nhat Khang, Nguyen Phan Tri Duc

Acknowledgement

We would like to express my appreciation to Assoc. Prof. Quan Thanh Tho for his invaluable guidance. Our research has greatly benefited from his deep knowledge, perceptive criticism, and constant support. Besides, we are grateful for the help from Mr. Nguyen Hieu Nghia from University of Information Technology - Vietnam National University (UIT - VNU) for valuable comments for research aspect and our proposed pipeline, including pointing out strengths as well as weaknesses of previous works on multimodal models so that we can elaborate on our ideas more effectively.

In addition, we would like to extend our gratitude to my family. Their unwavering faith in our abilities and constant encouragement have been my pillars of strength. Their belief in my potential has been a constant source of motivation and resilience. We are forever grateful for their love and support.

Abstract

This thesis explores the integration of multimodal deep learning models into search for fashion accessories, enhancing the user shopping experience through interactive capabilities. Our research focuses on developing a robust retrieval system that leverages both image and text inputs to provide accurate product suggestions. The system architecture allows users to input images of desired products along with additional textual descriptions to refine their search queries. Utilizing state-of-the-art multimodal language models, the system retrieves relevant product recommendations based on the similarity of visual and semantic features.

Key functionalities include providing detailed information about product availability, pricing, and suggesting alternative products based on specific customer preferences. The interactive nature of the system ensures a tailored shopping experience, enabling users to refine their search criteria and explore a diverse range of fashion accessories.

Preceded by comprehensive research on multimodal deep learning techniques frameworks, this thesis contributes to advancing the field of information retrieval. The experimental evaluation demonstrates the efficacy of the proposed approach in delivering accurate and relevant items, ultimately enhancing customer satisfaction and engagement in online shopping environments.

Table of Glossary

Term	Definition	Note
Multimodal	Relating to multiple modes of input/output, such as text and images, used in communication or computing systems.	
Chatbot	A computer program designed to simulate conversation with human users, typically over the internet.	
RAG System	Retrieval-Augmented Generation system, a framework that combines retrieval and generation models for enhanced natural language understanding and generation.	
CLIP Encoder	Contrastive Language-Image Pretraining Encoder, a deep learning model that learns joint representations of images and text through contrastive learning.	

Term	Definition	Note
End-to-end	A design process or system designed to operate without intermediate stages or interactions from separate components. In other words, it refers to the end-to-end nature of a process from start to finish without intermediate steps.	
Phase	A stage or step. For example, to solve a large problem, we would address smaller tasks (in phases). In the scope of the topic, there are phase 1 and phase 2.	
Transformer	A machine learning model architecture proposed in 2017, achieving high efficiency in Natural Language Processing (NLP).	Reference [49]
Pretraining	The process of training a machine learning model before it is fine-tuned for a specific task. In this stage, the model learns from large and diverse datasets to understand general patterns and representations, forming a strong semantic understanding. Pretraining is an important step in building high-performance models for various tasks.	

Term	Definition	Note
Fine-tuning	In machine learning, fine-tuning is the process of adjusting a model that has been pre-trained on a small or specific dataset. The goal is to make the model understand more and apply specific knowledge from new data or specific tasks.	

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In chapter 1, the overview, objectives, and goals of the research project are illustrated. The outline of the report is also presented.

1.1 Motivation

As the fashion market expands to cater to diverse demographics and preferences, the conventional methods of searching for products become increasingly inefficient. Consumers often face challenges in finding fashion items that align with their specific preferences, budget constraints, and other unique requirements. Traditional approaches rely heavily on human-assisted interactions through websites, where customers engage in conversations with store managers to seek guidance or recommendations. However, this manual process is time-consuming, prone to errors, and fails to leverage the full potential of technology in enhancing customer experiences. The inefficiency of this process raises a significant challenge: How can this whole process be automated to enhance customer experience and streamline operations for fashion stores?

By introducing a multimodal searching method, we aim to revolutionize the way customers interact with fashion stores online. The integration of image and text capabilities allows for a more intuitive and efficient communication channel between the consumer and the store. With this technology, customers can simply provide an image or description of the desired product, and the system can quickly analyze their preferences and recommend suitable options. Through fine-tuning a Vietnamese multimodal model

and implementing an efficient information retrieval pipeline, the system can understand customer preferences, recommend appropriate products, and provide personalized assistance, thus revolutionizing the way customers interact with fashion stores online.
Moreover, the development of such a system addresses the pressing need for automation in the fashion retail sector. As the number of fashion stores continues to rise, streamlining operations and improving customer experiences become paramount. A well-trained multimodal search system can effectively assist customers in navigating through the vast array of products, providing personalized recommendations, and answering queries regarding price, availability, and discounts.

1.2 Goals

The goals of this project encompass both technical advancements and practical outcomes aimed at revolutionizing the fashion retail experience. Firstly, the primary objective is to develop and deploy a sophisticated multimodal chatbot model capable of efficiently assisting customers in navigating the diverse offerings of fashion stores. This entails integrating image and text processing capabilities to accurately interpret customer requests and provide relevant product recommendations.
Additionally, a key goal is to enhance the overall customer experience by reducing the time and effort required for shopping online. By leveraging the chatbot's capabilities in understanding customer preferences and responding promptly to inquiries, the project aims to streamline the shopping process, making it more intuitive and convenient for users. Another crucial aspect of the project is to optimize the efficiency of fashion store operations through automation. This efficiency gain not only improves the productivity of fashion stores but also reduces operational costs and enhances overall business performance.
Moreover, the project seeks to foster innovation in the field of natural language processing and multimodal technology. By pushing the boundaries of what is achievable in terms of understanding and responding to customer queries, the development of the chatbot contributes to advancements in AI-driven conversational systems. This not only benefits the fashion industry but also has broader implications for other sectors seeking

to enhance customer interactions through automated solutions.

1.3 Scope

- Firstly, our project involves designing and implementing the core chatbot architecture that integrates both image and text processing capabilities. The chatbot should be able to accurately interpret customer inputs, whether they are in the form of images depicting desired products or text descriptions.
- Secondly, we will develop a model that can give most similar items to user based on user's input image and/or text prompt. Our baseline model to start this project will be CLIP [36], and we also propose a new model for encoding Vietnamese language along with the image, namely PhoCLIP.
- Thirdly, the project also involves implementing efficient information retrieval mechanisms to fetch product details such as price, availability, discounts, and other relevant information from the fashion store's database or external sources by Retrieval-Augmented Generation (RAG). This ensures that the chatbot can provide accurate and up-to-date information to customers in real-time.
- Finally, is our effort to design an intuitive and user-friendly interface for interacting with the chatbot is crucial for enhancing the overall user experience, by creating conversational flows, interactive elements, and visual representations to facilitate seamless communication between the chatbot and the user.

1.4 Thesis Structure

There are five chapters in our project:

- Chapter 1 provides a comprehensive introduction to the motivation behind the problem, its goals, objectives, and the scope of the project.
- Chapter 2 highlights previous research and related works pertaining to the task at hand, aiming to gain insights into existing methodologies and their limitations.

- Chapter 3 delves into the foundational concepts of Natural Language Processing (NLP), Vision models and Multimodal models, which are essential for the project's implementation.
- Chapter 4 is dedicated to our approach about the implementation details of our project, including the development of the multimodal chatbot and associated systems.
- Chapter 5 concludes our project by summarizing the key findings and outcomes, while also outlining our plans for future development and improvements.

Chapter 2

Related Works

The realm of multimodal artificial intelligence, which integrates visual and textual information, has witnessed significant advancements in recent years. This chapter provides an overview of key research works at the intersection of computer vision and natural language processing. From pioneering approaches such as utilizing convolutional neural network for ResNet, to cutting-edge architectures like the Vision Transformer and CLIP, this chapter surveys the landscape of multimodal understanding. Additionally, it explores emerging paradigms such as Visual Instruction Tuning and Retrieval-Augmented Generation model, highlighting the evolving methodologies in this field.

2.1 Vision Encoder

2.1.1 Vision Transformer

In recent years, the application of transformers, originally designed for natural language processing tasks, has been extended to the domain of computer vision, giving rise to the Vision Transformer (ViT) model. The seminal work by Dosovitskiy et al. introduced ViT as a powerful architecture for image recognition tasks. Departing from traditional Convolutional Neural Networks (CNN), ViT employs a self-attention mechanism to capture global dependencies within an image by treating it as a sequence of patches.

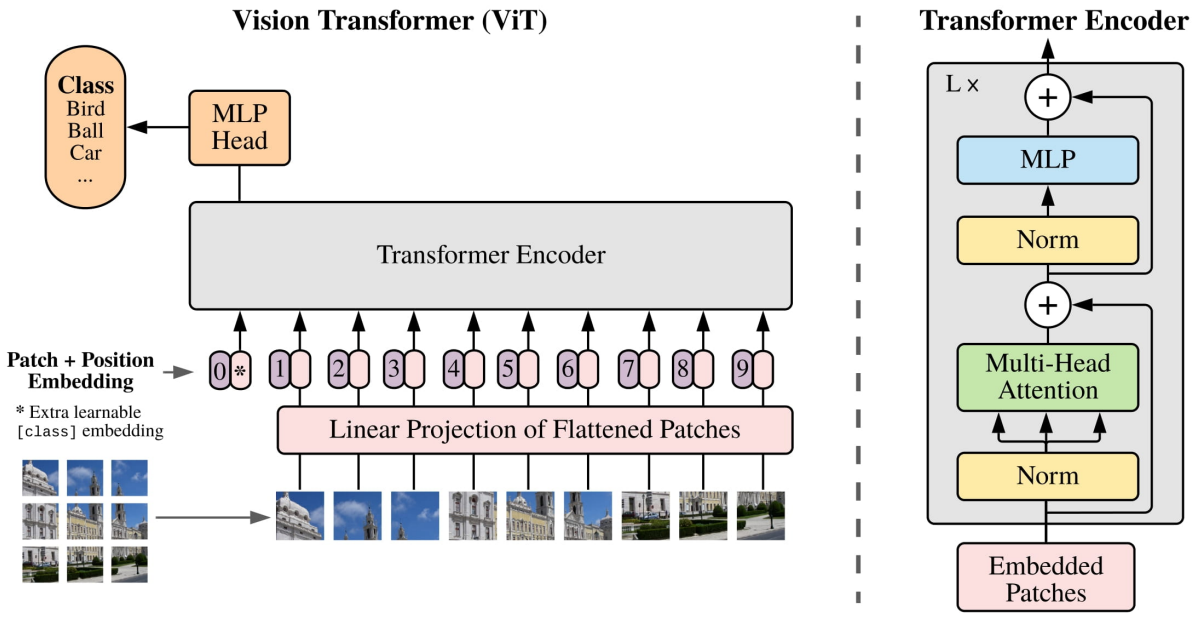


Figure 2.1: ViT architecture[1]

The core idea behind ViT is to represent images as sequences of fixed-size patches, which are then linearly embedded into high-dimensional feature vectors. These embeddings are then processed through a stack of transformer encoder layers, facilitating interaction between patches through self-attention mechanisms. This approach enables ViT to effectively model long-range dependencies and capture intricate visual patterns across the entire image.

One of the key advantages of ViT lies in its ability to handle images of arbitrary sizes, as opposed to CNNs which typically require fixed input dimensions. By leveraging self-attention mechanisms, ViT can effectively capture both local and global context from the image, making it robust to variations in scale and object layout.

Recent works such as CLIP, LLaVA [24] - [26], SigCLIP [56] explore novel techniques to improve the robustness and interpretability of vision transformers, paving the way for their broader adoption in real-world applications. Furthermore, ViT has demonstrated impressive performance on various benchmark datasets, rivaling or even surpassing the accuracy of conventional CNN-based architectures. This success has spurred further research into adapting transformer-based models for a wide range of computer vision tasks, including image classification, object detection, and image generation.

[1] `https://huggingface.co/docs/transformers/model_doc/vit`

2.1.2 Residual Neural Network

Traditional deep neural networks suffer from the vanishing gradient problem, where the gradients become increasingly small as they propagate through many layers during training. As a result, deeper networks often struggle to learn meaningful representations, leading to degradation in performance as the network depth increases.

Residual Neural Network (ResNet) tackles this issue by introducing skip connections, also known as shortcut connections or identity mappings, which enable the direct flow of information from one layer to another across multiple network depths. These skip connections allow the network to learn residual functions, i.e., the difference between the input and output of a layer, rather than trying to learn the entire mapping from input to output. This facilitates the training of much deeper networks without suffering from the vanishing gradient problem.

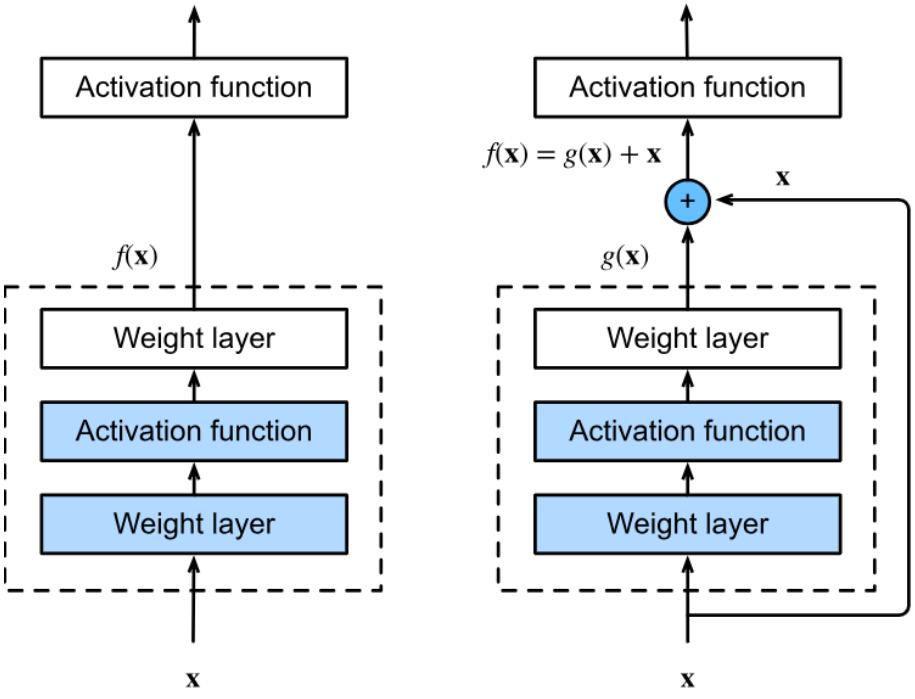


Figure 2.2: In a regular block (left), the portion within the dotted-line box must directly learn the mapping $f(x)$. In a residual block (right), the portion within the dotted-line box needs to learn the residual mapping $g(x) = f(x) - x$, making the identity mapping $f(x) = x$ easier to learn[1].

[1] https://www.geeksforgeeks.org/residual-networks-resnet-deep-learning/

The core idea behind ResNet can be encapsulated in the residual block, which comprises multiple convolutional layers with skip connections. These skip connections add the input to the output of the convolutional layers, effectively preserving the original information flow and enabling the network to learn residual mappings.
ResNet achieved unprecedented performance improvements on various computer vision tasks, including image classification, object detection, and image segmentation. Its architecture laid the foundation for subsequent advancements in deep learning research and served as the basis for numerous state-of-the-art models in computer vision.
While ResNet was a significant breakthrough in deep learning, several related works have built upon its foundations and extended its concepts to further improve model performance and efficiency.

- **Wide Residual Networks (WideResNet)**: Introduced by Zagoruyko and Komodakis in 2016, WideResNet extends the ResNet architecture by increasing the width of the network, i.e., the number of channels in each convolutional layer. This modification enhances the representational capacity of the network and improves its performance on various tasks.

- **ResNeXt**: Proposed by Xie et al. in 2017, ResNeXt introduces a novel architecture that emphasizes the importance of cardinality, i.e., the number of independent paths within a block. By leveraging grouped convolutions, ResNeXt achieves superior performance by effectively utilizing model capacity and computational resources.

- **DenseNet**: Introduced by Huang et al. in 2017, DenseNet proposes a densely connected architecture where each layer receives input from all preceding layers. This dense connectivity facilitates feature reuse and enhances gradient flow, leading to improved parameter efficiency and performance.

2.2 Language Models

Language models play a crucial role in natural language processing tasks, ranging from text generation and machine translation to sentiment analysis and named entity recognition. Over the years, several language models have been proposed, each with its own

strengths and limitations.

2.2.1 Traditional Language Models

Traditional language models, such as n-gram models and Probabilistic Context-Free Grammars (PCFGs), have been widely used in natural language processing tasks. These models rely on simple statistical methods and handcrafted features to generate or predict sequences of words. While effective for some tasks, they often struggle with capturing long-range dependencies and semantic nuances in language.
Traditional language models relied on various methods to understand and generate language, including:

- **n-gram models**: These models estimate the probability of a word based on the previous $(n-1)$ words. While computationally efficient, they lack a deep understanding of language structure and context.
- **Hidden Markov models**: They model sequences of data, including text, using hidden states and observable states. They rely on transition probabilities between hidden states and emission probabilities for observable states.
- **Rule-based systems**: These systems utilize predefined rules for language processing and generation. They incorporate linguistic analysis, syntactic rules, and domain-specific knowledge.
- **Part-of-Speech tagging**: This involves labeling each word in a sentence with its grammatical part of speech, such as noun, verb, or adjective. It is useful for various NLP tasks.
- **Syntactic and semantic parsing**: These models parse sentences into structured representations, capturing syntactic and semantic relationships between words. They are used for tasks like information extraction and question answering.
- **Statistical language models**: These models estimate the likelihood of a word given its context using statistical techniques such as n-grams, conditional probabilities, and smoothing.

- **Statistical machine translation**: They models translate text between languages using statistical methods learned from parallel corpora. However, they have been largely replaced by Neural Machine Translation models.

2.2.2 Neural Language Models

In recent years, neural language models have gained prominence due to their ability to capture complex patterns and dependencies in text data. These models are typically based on neural networks, such as RNNs, LSTMs, and transformer architectures.

Among neural language models, the Transformer architecture introduced in the ”Attention is All You Need” paper by Vaswani et al. has been particularly influential. Transformers have achieved state-of-the-art performance in various natural language processing tasks, including language modeling, machine translation, and text classification.

2.2.3 PhoBERT: Pre-trained language models for Vietnamese

PhoBERT is a pre-trained language model specifically designed for the Vietnamese language. It is based on the Transformer architecture and trained on a large corpus of Vietnamese text data. PhoBERT is developed by VinAI Research and has been fine-tuned for various downstream tasks, including sentiment analysis, named entity recognition, and part-of-speech tagging.

PhoBERT extends the Transformer architecture and is specifically tailored for the Vietnamese language. It consists of multiple layers of self-attention and feed-forward networks, similar to the original Transformer architecture. However, PhoBERT is pre-trained on a large corpus of Vietnamese text data, allowing it to learn contextual representations of Vietnamese words and phrases.

In addition to the standard Transformer layers, PhoBERT incorporates several enhancements to better handle the characteristics of the Vietnamese language. These enhancements may include modifications to the tokenization process, the addition of language-specific features, and fine-tuning of hyperparameters to optimize performance on Vietnamese language tasks.

PhoBERT comes in two versions: $\text{PhoBERT}_{\text{base}}$ and $\text{PhoBERT}_{\text{large}}$, which utilize the same architectures as $\text{BERT}_{\text{base}}$ and $\text{BERT}_{\text{large}}$, respectively. The pre-training approach of

PhoBERT is based on RoBERTa [27], which optimizes the BERT pre-training procedure for more robust performance.

Pre-training Data

A 20GB pre-training dataset of uncompressed texts is used. This dataset is a concatenation of two corpora: the Vietnamese Wikipedia corpus (1GB) and a 19GB corpus generated by removing similar articles and duplication from a 50GB Vietnamese news corpus crawled from a wide range of news websites and topics.
The RDRSegmenter [29] from VnCoreNLP [50] is employed to perform word and sentence segmentation on the pre-training dataset. This results in 145 million word-segmented sentences (3 billion word tokens). Subsequently, fastBPE (Sennrich, Haddow, and Birch) is applied to segment these sentences with subword units, using a vocabulary of 64K subword types. On average, there are 24.4 subword tokens per sentence.

Optimization

The RoBERTa implementation in fairseq (Ott et al.) is employed for optimization. A maximum length of 256 subword tokens is set, generating $145\text{M} \times 24.4/256 \approx 13.8\text{M}$ sentence blocks. The models are optimized using Adam [21]. A batch size of 1024 across 4 V100 GPUs (16GB each) and a peak learning rate of 0.0004 are used for $\text{PhoBERT}_{\text{base}}$, while for $\text{PhoBERT}_{\text{large}}$, a batch size of 512 and a peak learning rate of 0.0002 are used. The pre-training process runs for 40 epochs, with the learning rate warmed up for 2 epochs, resulting in approximately 540K training steps for $\text{PhoBERT}_{\text{base}}$ and 1.08M training steps for $\text{PhoBERT}_{\text{large}}$. $\text{PhoBERT}_{\text{base}}$ is pre-trained for 3 weeks, followed by $\text{PhoBERT}_{\text{large}}$ pre-training for 5 weeks.

2.2.4 Other Large Language Models

ChatGPT

ChatGPT, developed by OpenAI, builds upon the success of previous models like InstructGPT, incorporating techniques like Reinforcement Learning from Human Feedback (RLHF) to provide coherent and interpretative responses. It has been a prominent

trend since early 2023, and the recent announcement of GPT-4, an upgrade from the GPT3.5 base of ChatGPT, signifies continued development in this line.

LLaMa-2, PaLM, Mistral

LLaMa-2, PaLM, and Mistral are open-source Large Language Models (LLMs) provided by Meta and Google. They aim to empower the community with accessible language models for research and development, contributing to the advancement of LLM technology.

Vietnamese Market

In the Vietnamese market, models like URA-LLaMa, PhoGPT, and Bloomz are gaining traction. While these models and their underlying technologies are relatively new, efforts are underway to expand the Vietnamese-language LLM ecosystem. Although the current contributions to open-source Vietnamese LLMs are limited, there is a focus on utilizing these models to train and enhance knowledge in the Vietnamese language. These LLMs can be employed for question-answering tasks based on provided contextual information. Moreover, recent advancements have introduced BARTpho [47] and ViT5 [35] to the Vietnamese NLP landscape. BARTpho, a variant of BART fine-tuned specifically for Vietnamese, offers powerful text generation capabilities tailored to the intricacies of the Vietnamese language. On the other hand, ViT5, based on the ViT architecture, excels at processing visual information and has been adapted to handle Vietnamese text as well. These models require substantial computational resources for training and inference due to their complexity and large parameter sizes.

Considering the scope of our project, BARTpho emerges as a suitable choice for initial experimentation. Its fine-tuned nature for the Vietnamese language and robust text generation abilities align well with our objectives for developing and testing our inferencing pipeline. By leveraging BARTpho as a starting point, we can effectively explore and refine our approach within the Vietnamese market context.

2.3 Retrieval-Augmented Generation

2.3.1 Information Retrieval

Recent advancements in information retrieval have been driven by the integration of dense representations and neural models, which aim to overcome the limitations of traditional sparse representations and term-matching techniques. Traditionally, information retrieval methods relied on approaches like TF-IDF [38] (Term Frequency - Inverse Document Frequency) and BM25 [1] (Best Match 25) to rank documents based on their similarity to the query. However, these methods often struggled to capture semantic and contextual information, resulting in challenges retrieving relevant documents that did not share common words with the query.

In response to these limitations, recent research has explored the application of neural architectures based on pre-trained transformer models, such as BERT [9], to enhance information retrieval. Three prominent neural architectures have emerged: Bi-encoder, cross-encoder, and poly-encoder models [17]. Bi-encoder architectures, which encode documents independently and compare them using cosine similarity, offer efficiency and scalability in indexing and retrieval tasks. However, they may lack the ability to capture fine-grained interactions between queries and documents.

Cross-encoder models, on the other hand, concatenate queries and documents and process them together to produce relevance scores. This approach allows for full attention over both the query and the document, resulting in higher accuracy and expressiveness [31]. However, cross-encoder models tend to be slower and computationally intensive, as they require re-encoding for each query-document pair. To address this challenge, poly-encoder models have been introduced, leveraging separate encoders for queries and documents and applying attention only at the top layer. This design achieves better performance than bi-encoder models while maintaining faster processing speeds than cross-encoder models.

2.3.2 Generative Models

Generative models represent a category of statistical models capable of producing new datasets based on estimated probability distributions, thereby generating samples resembling the original data. Unlike discriminative models, generative models possess the ability to generate novel samples without the need for labeled data, making them valuable in unsupervised learning scenarios where labeled data may be scarce or unavailable [14]. Among the most prominent generative models are Variational Auto-Encoders (VAEs) and Generative Adversarial Networks (GANs), which have gained significant attention and application across various domains.

VAEs consist of an encoder, decoder, and a loss function, typically comprising input, middle (or bottleneck), and output layers. The encoder encodes input data into a lower-dimensional representation stored in the middle layer, while the decoder reconstructs the input from this representation. VAEs leverage two loss functions - reconstruction loss and Kulback-Leiber divergence - to align the latent space with assumed distributions, allowing for efficient representation learning and generation of new samples [3].

In contrast, GANs operate uniquely in the domain of image datasets, excelling in generating realistic images. GANs consist of two complementary components: A generator and a discriminator. The generator creates fake data by incorporating feedback from the discriminator, which aims to classify its output as real. Through adversarial training, the discriminator learns to distinguish between real and generated samples, while the generator continually improves its ability to produce realistic data [3].

While generative models have seen limited application in web quality assessment and modeling, they hold promise in various domains, particularly in underwater image restoration and enhancement. In marine research, GANs have emerged as a breakthrough technique for addressing challenges such as turbidity and color distortion in underwater observation. Researchers have developed GAN models specifically for underwater image restoration, leveraging techniques like cycle-consistent adversarial networks to enhance contrast and correct color distortion. However, challenges remain, particularly in scenarios with inhomogeneous illumination, where existing models may struggle to maintain performance [28]. Despite these challenges, ongoing research efforts are exploring various GAN-based deep learning architectures for underwater image enhancement, includ-

ing encoder–decoder models, block designs, dual-generator GANs, and multi-branch designs, offering promising avenues for further advancements in this field [2].

2.3.3 Combining Retriever with Generative Models

Retriever models primarily focus on retrieving relevant information from large corpora or knowledge bases based on a given query or context. These models employ techniques like sparse representations and term-matching to efficiently retrieve documents or passages containing relevant information. However, retriever models may struggle to capture semantic and contextual nuances, leading to limitations in understanding complex queries or generating diverse responses.

On the other hand, generative models, such as transformer-based language models like GPT, excel at generating coherent and contextually relevant text. These models leverage large-scale pre-training on diverse text data to learn rich representations of language and produce human-like responses to input prompts. However, generative models may lack the ability to retrieve specific factual information or leverage external knowledge effectively.

The idea of unifying retriever models with generative models aims to combine the strengths of both approaches to create more powerful and versatile language models. By integrating retriever components into generative architectures, models can effectively retrieve relevant information from external sources or knowledge bases and incorporate it into the generation process. This integration enables the model to produce responses that are not only contextually relevant and coherent but also factually accurate and informed by external knowledge.

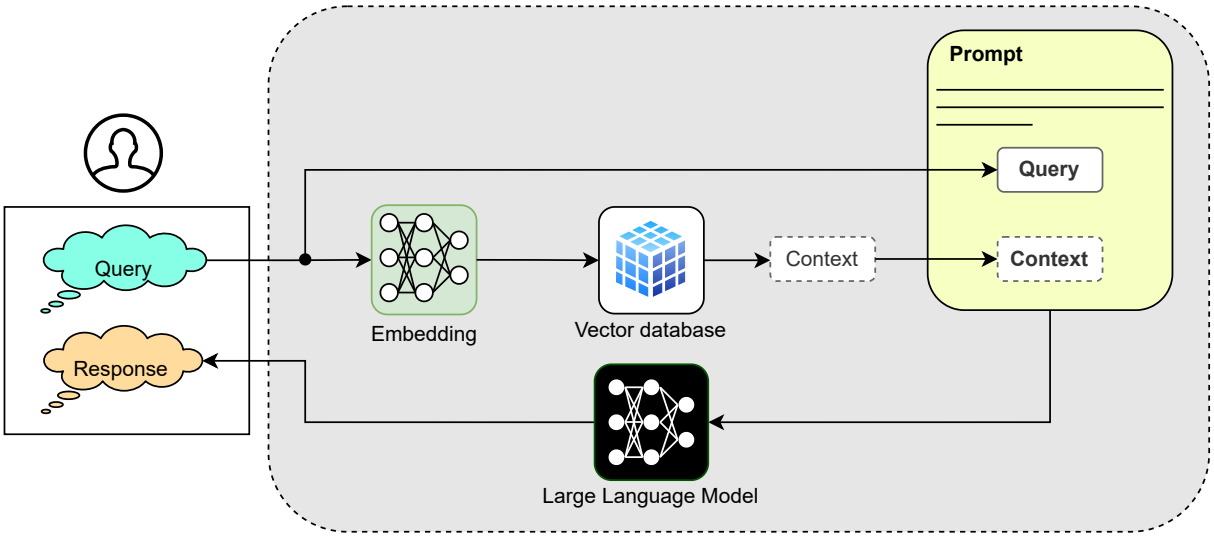


Figure 2.3: Retrieval-Augmented Generation pipeline

Several approaches have been proposed to unify retriever models with generative models. Retrieval-Augmented Generation (RAG) incorporate retriever components to retrieve relevant passages from a knowledge base, which are then used to enhance the generation process of transformer-based language models. Lewis et al. combines a Pre-trained Language Model (PLM) like BART with a Dense Passage Retriever (DPR) using the Bi-encoder architecture in order to utilize input sequences to retrieve pertinent passages from resources like Wikipedia and integrates them as additional context during generation. Overall, the unification of retriever models with generative models represents a promising direction in NLP research, offering the potential to create more robust and informative language models that can effectively understand, generate, and utilize information from diverse sources.

2.3.4 Combining Multimodal Model with Vector Database

Traditionally, information retrieval systems have relied on vector databases to store and search for multimedia content. Vector databases excel at efficient retrieval based on similarity, but they struggle to capture the nuances and context inherent in multimedia data. Here, multimodal LLMs emerge as a powerful complement, offering high-fidelity descriptions and the ability to understand subjective terms and emotional cues.
This combination offers several advantages. It overcomes the limitations of vector databases' context-free representations. Multimodal LLMs can generate rich descriptions that cap-

ture the scene, interactions, and sentiment conveyed in images or the tone and underlying implications in audio. This allows for more accurate and insightful searches. The integration empowers a system to interact with users intelligently. Overall, Vector databases often require additional layers of complexity, including vector maintenance and model updates. A unified platform that leverages LLMs can streamline data storage, search, analysis, and content generation across various modalities. This not only reduces operational costs but also eliminates the need for managing multiple complex systems.

2.4 Multimodal Image-Text Contrastive Learning

Contrastive Learning is a technique utilized to learn a representation of data, ensuring that similar instances are clustered closely together in the representation space, while dissimilar instances are pushed farther apart.

Multimodal Image-Text Contrastive Learning extends this principle to learn joint representations of both images and text, enabling the model to understand the relationships between visual and textual modalities. By training on pairs of corresponding images and text descriptions, the model learns to map them into a shared embedding space where semantically similar pairs are closer together, while dissimilar pairs are separated.

OpenAI's CLIP [36] stands out as a pioneering model in this domain. It effectively learns a joint representation of images and text by leveraging large-scale datasets. Through this training process, CLIP can associate images and text within a multimodal embedding space. Consequently, it excels in various tasks such as image classification, zero-shot image classification, and text-based image retrieval. The architecture depicted in Figure 2.4 illustrates the CLIP system's architecture.

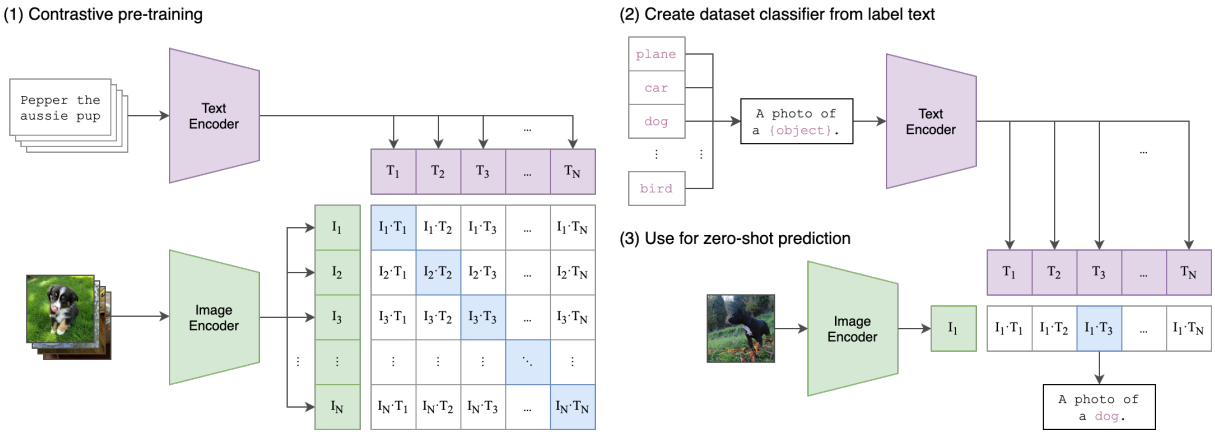


Figure 2.4: CLIP jointly trains an image encoder and a text encoder to predict the correct pairings of a batch of (image, text) training examples. At test time the learned text encoder synthesizes a zero-shot linear classifier by embedding the names or descriptions of the target dataset's classes.

LiT [55], or Locked-image Text Tuning, is another noteworthy technique designed to enhance CLIP's capabilities. The idea behind the contrastive pre-training approach is to learn two embedding models: an image model and a text model, both of which produce representations of the same dimensionality. These models are trained using a contrastive loss. Contrastive pre-training can be viewed as learning two tasks at the same time: learning an image embedding and learning a text embedding to align with the image embedding space.

Introducing pre-trained image or text models into the contrastive learning setting involves several design choices. First, each tower (image and text) can independently be initialized randomly or from a pre-trained model. For a pre-trained model there are at least two variants: we can lock (freeze) it or allow fine-tuning. For LiT, the key idea is to tune the text tower using image-text data, while using a pre-trained, strong image model as the image tower, which just teaches a text model to read out suitable representations from a pre-trained image model. This approach facilitates efficient adaptation to various downstream tasks without the need for extensive retraining.

Chinese-CLIP [52] is a specialized variant tailored explicitly for the Chinese language and culture. They use the same architecture as OpenAI CLIP, initialize the image encoder with the OpenAI CLIP models, and the text encoder with the Chinese RoBERTa models. In Stage 1, they freeze the image encoder and only optimize the text encoder with LiT, and

in Stage 2, both encoders are trained with contrastive tuning. In this way, the new model can inherit from the foundation models through initialization and LiT, and effectively transfer to language-specific data through contrastive tuning.

Trained on Chinese text and image data, Chinese-CLIP aims to provide precise and culturally relevant image-text representations for Chinese-speaking users. This variant opens up new opportunities for applications in image search, recommendation systems, and more.

2.5 Discussion

The integration of visual and textual information in multimodal artificial intelligence has seen remarkable progress, evidenced by a plethora of innovative research works at the intersection of computer vision and natural language processing. In this discussion, we delve into the strengths and highlights of multimodal techniques, particularly focusing on vision models, while also addressing their limitations and potential challenges.

2.5.1 Strengths and Highlights

One of the key strengths of multimodal techniques lies in their ability to harness the complementary nature of visual and textual modalities, enabling more comprehensive understanding and interpretation of complex data. Dosovitskiy et al. have demonstrated the efficacy of leveraging transformer architectures originally designed for natural language processing tasks in the domain of computer vision. By treating images as sequences of patches and applying self-attention mechanisms, ViT models excel at capturing both local and global dependencies within images, achieving impressive performance on various benchmark datasets.

CLIP has demonstrated remarkable capabilities in understanding and associating images with text across various languages and cultures. Leveraging its multimodal contrastive learning approach, CLIP can learn joint representations of images and text, enabling it to excel in tasks such as image classification and text-based image retrieval. One of the strengths of CLIP's architecture lies in its adaptability, while the original CLIP model was trained English, its architecture allows for straightforward integration with

language-specific text encoders.

Specifically, for the Vietnamese language, we can harness the power of CLIP by integrating it with PhoBERT, a state-of-the-art text encoder trained specifically for Vietnamese. By replacing the original text encoder in CLIP with PhoBERT, we can create a specialized variant tailored for understanding Vietnamese text and its association with images. Building upon the principles of LiT techniques, we can further enhance the performance of CLIP with PhoBERT by fine-tuning the text encoder using image-text data. This approach not only ensures that the model captures the nuances of the Vietnamese language but also aligns its representations with the visual semantics captured by CLIP's image encoder.

By combining the strengths of CLIP's multimodal representation learning with the language-specific capabilities of PhoBERT, we can create a powerful framework for understanding and interpreting multimodal data in the context of Vietnamese culture and language. This has wide-ranging implications for applications such as image search, recommendation systems, and cultural analysis tailored to Vietnamese-speaking users.

2.5.2 Limitations and Challenges

While multimodal techniques hold promise for a wide range of applications, they also face several limitations and challenges that merit attention.

One significant challenge is the lack of benchmarks and resources tailored specifically for languages such as Vietnamese. Most existing benchmark datasets and evaluation metrics are designed for widely spoken languages like English, making it difficult to assess the performance of multimodal models in languages with fewer resources. This limitation hampers the development and evaluation of multimodal systems for Vietnamese-speaking users and highlights the need for dedicated efforts to create comprehensive benchmarks and evaluation protocols for languages with smaller speaker populations.

Another limitation stems from the current state of multimodal model architectures, such as CLIP, which may not natively support languages like Vietnamese. While CLIP has demonstrated impressive capabilities in understanding and associating images with text in languages it was trained on, extending its functionality to languages beyond its pre-training data presents challenges. Integrating language-specific text encoders like PhoBERT

with CLIP is a promising approach, but it requires careful consideration of compatibility, performance, and fine-tuning strategies to ensure effective multimodal representation learning for languages like Vietnamese.

While original multimodal models, such as CLIP, excel at understanding the relationship between text and images, they primarily generate textual outputs. This limitation inhibits their direct application in tasks requiring image generation or manipulation. One intriguing solution to this limitation is the development of hybrid pipelines that combine multimodal models with techniques like RAG. By integrating multimodal understanding with generation capabilities, such hybrid pipelines could potentially generate images based on textual prompts, offering a more comprehensive approach to multimodal tasks.

The lack of benchmarks and resources tailored specifically for languages such as Vietnamese. Most existing benchmark datasets and evaluation metrics are designed for widely spoken languages like English, making it difficult to assess the performance of multimodal models in languages with fewer resources. This limitation hampers the development and evaluation of multimodal systems for Vietnamese-speaking users and highlights the need for dedicated efforts to create comprehensive benchmarks and evaluation protocols for languages with smaller speaker populations.

Chapter 3

Theoretical Background

This chapter presents preliminary knowledge which is fundamental of our project.

3.1 Stochastic Gradient Descent

Stochastic Gradient Descent (SGD) is a popular optimization algorithm used in training machine learning models, particularly in scenarios where the dataset is large. It is an iterative optimization algorithm that aims to minimize the loss function by adjusting the model parameters in the direction that reduces the loss.

Consider a machine learning model with parameters represented by θ. The goal is to minimize a loss function $L(\theta)$, which measures the difference between the predicted output of the model and the actual target values. In the context of supervised learning, this loss function typically quantifies the error between predictions and ground truth. SGD is used for updating the parameters θ in small steps by computing the gradient of the loss function with respect to θ for a randomly selected subset of the training data. This subset is often referred to as a mini-batch, and its size is denoted by b. The update rule for SGD can be expressed as:

$$\theta_{t+1} = \theta_t - \eta \cdot \nabla_\theta L(\theta_t; x_{i:i+b}, y_{i:i+b}) \tag{3.1}$$

where:

- θ_t is the parameter vector at iteration t.

- η is the learning rate, which determines the step size for parameter updates.
- $\nabla_\theta L(\theta_t; x_{i:i+b}, y_{i:i+b})$ is the gradient of the loss function L with respect to the parameters θ, computed using a mini-batch of training data indexed from i to $i + b$.

In each iteration of SGD, a mini-batch of training examples is randomly sampled from the dataset, and the gradient of the loss function is computed with respect to the parameters using this mini-batch. The parameters are then updated in the direction opposite to the gradient, scaled by the learning rate η.
By iteratively applying this update rule over the entire dataset for multiple epochs, SGD gradually converges towards a set of parameters that minimize the loss function.

3.2 Multi-Layer Perceptron

3.2.1 Layer

Layers in an Multi-Layer Perceptron (MLP) are akin to stacked computational units, each contributing uniquely to the network's ability to transform input data into meaningful output.

- **Input Layer:** At the forefront of the network lies the input layer, where data is ingested. This layer comprises nodes, each representing a feature or dimension of the input data. The input layer essentially acts as the interface between the external environment and the neural network. Usually, we denote $x_i^{(0)}$ as the value of the i^{th} node in the input layer, where $i = 1, 2, ..., n^{(0)}$, and $n^{(0)}$ is the number of nodes in the input layer.
- **Hidden Layers:** Between the input and output layers are one or more hidden layers. These layers are where the magic happens – where the network learns intricate patterns and relationships within the data. Each node in a hidden layer receives inputs from the previous layer, performs a computation (typically a weighted sum followed by an activation function), and passes the result to the next layer. We denote $x_j^{(l)}$ as the value of the j^{th} node in the l^{th} hidden layer, where $l = 1, 2, ..., L$, and $j = 1, 2, ..., n^{(l)}$, and $n^{(l)}$ is the number of nodes in the l^{th} hidden layer.

- **Output Layer:** The transformed data emerges from the output layer. The structure and function of this layer depend on the nature of the task the MLP is designed to solve. For instance, in a classification task, the output layer might consist of nodes representing different classes, with each node outputting a probability score indicating the likelihood of the input belonging to that class. Let $y_k^{(L)}$ denotes the value of the k^{th} node in the output layer, where $k = 1, 2, ..., n^{(L)}$, and $n^{(L)}$ is the number of nodes in the output layer.

Figure 3.1 demonstrates a MLP architecture with 2 hidden layers.

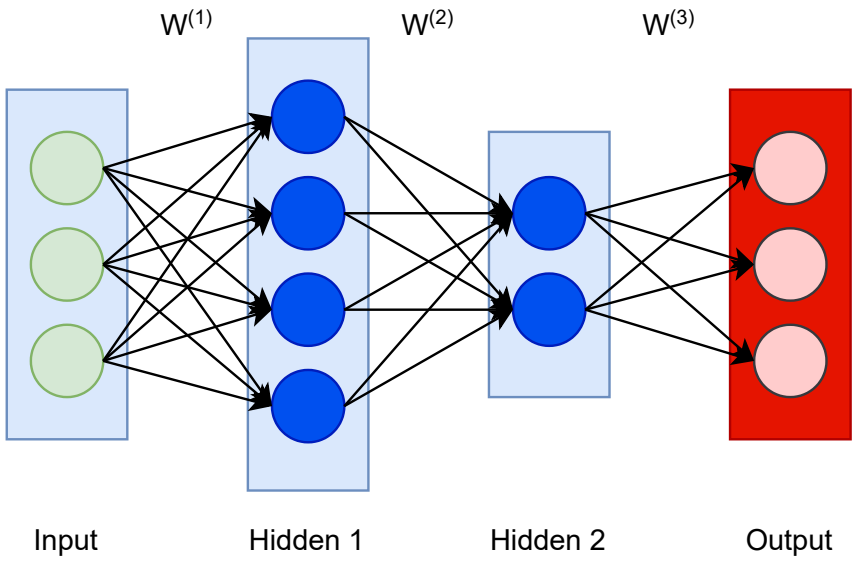


Figure 3.1: MLP architecture with 2 hidden layers

The layers in an MLP are interconnected via weighted connections, which serve as the conduits for information flow and learning. Through a process known as backpropagation, the network adjusts these weights iteratively during training to minimize the discrepancy between its predictions and the actual target values. The depth of an MLP – determined by the number of hidden layers – is a crucial factor influencing its capacity to learn complex relationships within data. Deeper architectures allow for hierarchical representations of features, enabling the network to capture increasingly abstract and nuanced patterns.

3.2.2 Node (Unit)

Nodes are the elemental units within a neural network that process and transmit information. Also referred to as neurons or units, nodes serve as the building blocks of the network's computational capabilities.

- **Input Nodes:** The input layer of an MLP consists of input nodes, each corresponding to a feature or dimension of the input data. These nodes serve as the entry point for external data into the network. Input nodes simply pass along the values of the input features without any computation.
- **Hidden Nodes:** Hidden nodes reside within the hidden layers of the MLP and are responsible for processing the incoming information from the preceding layer. Each hidden node receives inputs from all the nodes in the previous layer, computes a weighted sum of these inputs, applies an activation function, and forwards the result to the next layer. The collective activity of hidden nodes enables the network to learn and represent complex patterns and relationships within the data.
- **Output Nodes:** At the output layer, output nodes generate the final predictions or outputs of the network. The number of output nodes typically corresponds to the number of distinct classes or targets in a classification or regression task, respectively. Each output node produces a value or a probability score representing the network's confidence in the prediction for a specific class or target.

Nodes within an MLP are characterized by their activation functions, which introduce non-linearity into the network and enable it to approximate complex functions. Common activation functions include the sigmoid function, hyperbolic tangent (tanh) function, and rectified linear unit (ReLU) function, among others. The choice of activation function impacts the network's ability to learn and generalize from the training data.
During the training process, nodes undergo parameter learning, wherein the network adjusts the weights associated with each connection and the biases associated with each node to minimize the discrepancy between its predictions and the actual target values. This iterative optimization process, typically facilitated by backpropagation and gradient descent algorithms, enables the network to improve its performance over time.

3.2.3 Weight and Bias

In a neural network, weights (W) and biases (b) are parameters that are learned during the training process. They play a crucial role in determining the behavior and performance of the network.

Weights

Weights represent the strength of connections between neurons in adjacent layers of the network. Let's denote the weight connecting the i^{th} neuron in the l^{th} layer to the j^{th} neuron in the $(l+1)^{th}$ layer as $W_{ij}^{(l)}$. Thus, the weight matrix $W^{(l)}$ for the l^{th} layer can be represented as:

$$W^{(l)} = \begin{bmatrix} W_{11}^{(l)} & W_{12}^{(l)} & \cdots & W_{1m}^{(l)} \\ W_{21}^{(l)} & W_{22}^{(l)} & \cdots & W_{2m}^{(l)} \\ \vdots & \vdots & \ddots & \vdots \\ W_{n1}^{(l)} & W_{n2}^{(l)} & \cdots & W_{nm}^{(l)} \end{bmatrix}$$

where:

- n is the number of neurons in the l^{th} layer.
- m is the number of neurons in the $(l+1)^{th}$ layer.

During training, these weights are adjusted through techniques like backpropagation and gradient descent to minimize the error between the predicted output and the actual output.

Biases

Biases are additional parameters in neurons that allow them to shift the activation function. Let's denote the bias of the j^{th} neuron in the $(l+1)^{th}$ layer as $b_j^{(l+1)}$. Thus, the bias vector $b^{(l+1)}$ for the $(l+1)^{th}$ layer can be represented as:

$$b^{(l+1)} = \begin{bmatrix} b_1^{(l+1)} \\ b_2^{(l+1)} \\ \vdots \\ b_m^{(l+1)} \end{bmatrix}$$

During the forward pass of the network, biases are added to the weighted sum of inputs before passing through the activation function.

3.2.4 Activation Functions

Sigmoid

The sigmoid function, also known as the logistic function, is a popular activation function used in neural networks. It is defined mathematically as:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

where z is the input to the function.
The sigmoid function maps any real-valued number to the range $(0, 1)$, making it particularly useful in binary classification tasks where the output needs to be interpreted as a probability. Key properties of the sigmoid function are:

- **S-shaped curve:** The sigmoid function produces an S-shaped curve, which means that small changes in the input z result in small changes in the output $\sigma(z)$ when $\sigma(z)$ is close to the extremes (0 or 1), and larger changes when $\sigma(z)$ is close to 0.5.
- **Differentiability:** The sigmoid function is differentiable everywhere, allowing for the use of gradient-based optimization algorithms such as gradient descent during training.
- **Monotonicity:** The sigmoid function is monotonically increasing, meaning that as the input z increases, the output $\sigma(z)$ also increases, and vice versa.
- **Range:** The output of the sigmoid function is always bounded between 0 and 1, which can help stabilize training and prevent the activation from saturating (i.e., reaching 0 or 1) too quickly.
- **Centering around zero:** When the input z is close to zero, the sigmoid function outputs approximately 0.5, making it convenient for initializing weights in neural networks.

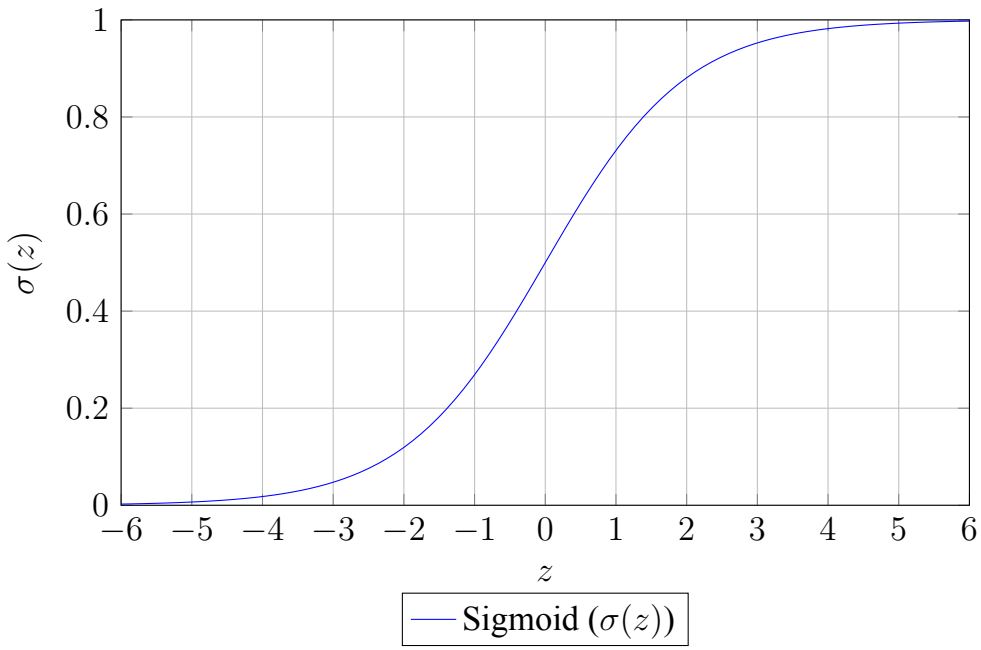


Figure 3.2: Graph of the sigmoid function.

Despite its popularity, the sigmoid function has some drawbacks, such as vanishing gradients and the tendency to saturate when dealing with very positive or very negative inputs, which can slow down learning in deep networks.

Tanh

The hyperbolic tangent (tanh) function is another commonly used activation function in neural networks. It is defined mathematically as:

$$\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

where z is the input to the function.

The tanh function is similar to the sigmoid function but maps input values to the range $(-1, 1)$. This property allows the tanh function to be zero-centered, which can aid in optimizing the weights during training. Some properties of the tanh function are:

- **S-shaped Curve:** Similar to the sigmoid function, the tanh function produces an S-shaped curve, which means that small changes in the input z result in small changes in the output $\tanh(z)$ when $\tanh(z)$ is close to the extremes (-1 or 1), and larger changes when $\tanh(z)$ is close to 0.

- **Differentiability:** The tanh function is differentiable everywhere, allowing for the use of gradient-based optimization algorithms such as gradient descent during training.
- **Zero-centered:** The output of the tanh function is centered around zero, which can help stabilize training by preventing the gradients from becoming too large or too small.
- **Range:** The output of the tanh function is always bounded between -1 and 1, making it suitable for networks where the input values may be negative or positive.

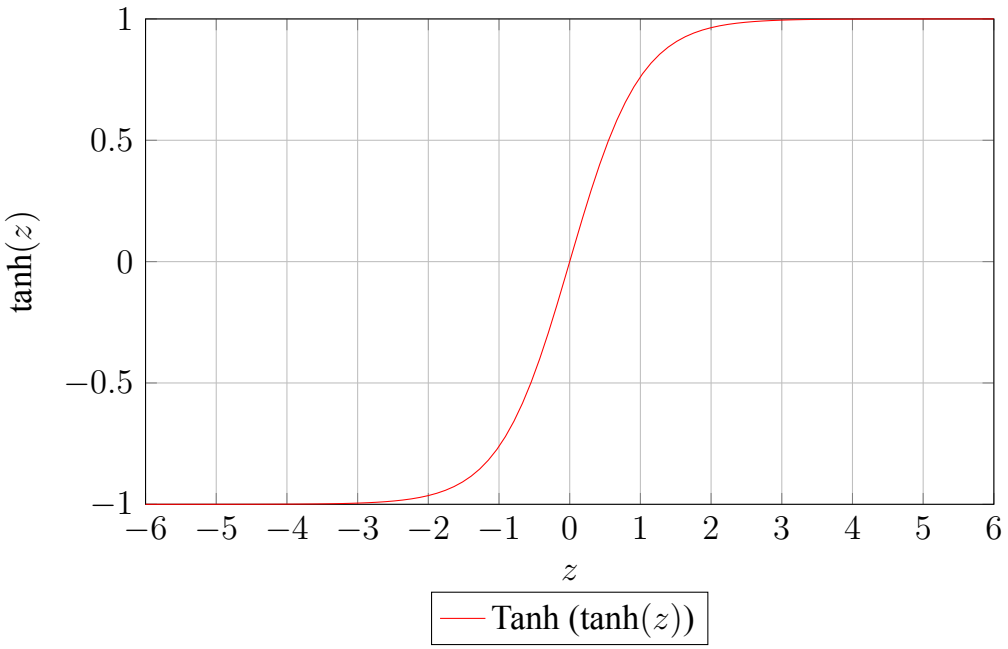


Figure 3.3: Graph of the tanh function.

ReLU

The Rectified Linear Unit (ReLU) function is a popular activation function in neural networks, especially in deep learning architectures. It is defined mathematically as:

$$\text{ReLU}(z) = \max(0, z)$$

where z is the input to the function.

The ReLU function outputs the input value if it is positive, and zero otherwise. This simple thresholding operation introduces non-linearity to the network, allowing it to learn complex relationships in the data. Some properties of the ReLU function include:

- **Sparsity:** The ReLU function produces sparse representations by setting negative values to zero. This sparsity can help reduce overfitting by preventing the network from learning redundant features.
- **Efficiency:** ReLU is computationally efficient compared to other activation functions like sigmoid and tanh, as it involves only simple thresholding operations.
- **Avoiding vanishing gradients:** Unlike sigmoid and tanh functions, ReLU does not suffer from the vanishing gradient problem for positive inputs, which can accelerate training in deep networks.
- **Non-saturating:** ReLU does not saturate for positive inputs, allowing the network to learn quickly and efficiently.

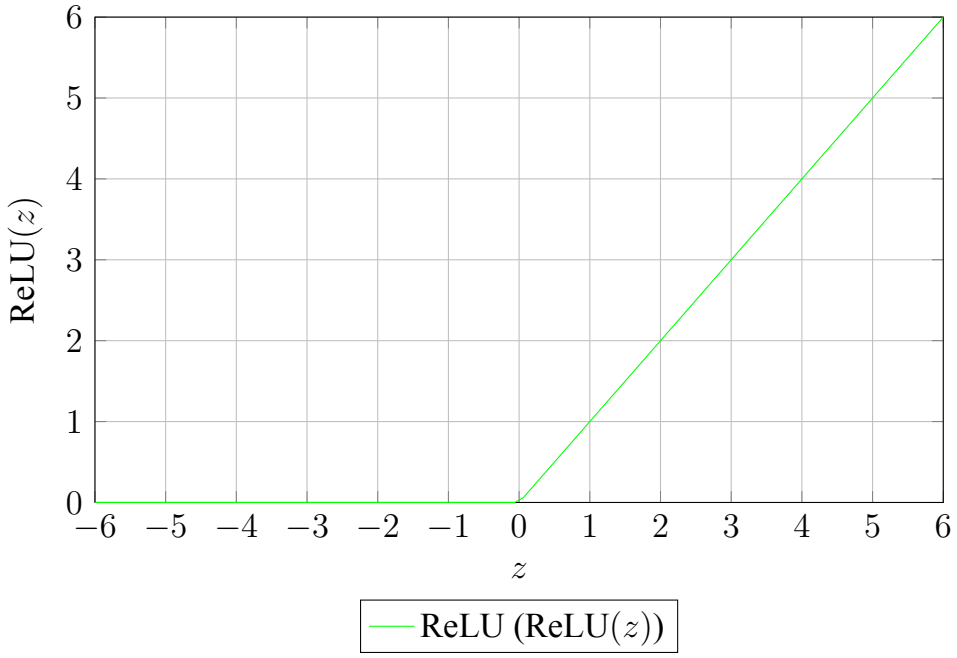


Figure 3.4: Graph of the ReLU function.

Softmax

The softmax function is commonly used in neural networks for multi-class classification tasks. It takes a vector of real-valued numbers as input and normalizes them into a probability distribution. The softmax function is defined mathematically as:

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

where z_i is the i^{th} element of the input vector **z**, and K is the number of classes. Key properties of the softmax function include:

- **Output as probabilities:** The softmax function transforms the input vector into a probability distribution over K classes, where each element represents the probability of the corresponding class.
- **Normalization:** The denominator in the softmax function ensures that the output probabilities sum to 1, making it suitable for multi-class classification tasks.
- **Sensitivity to input differences:** The softmax function amplifies differences between the input values, which can help the network make more confident predictions.
- **Differentiability:** The softmax function is differentiable, enabling the use of gradient-based optimization techniques for training neural networks.

3.2.5 Backpropagation

Foundation

The most popular method for optimizing MLP networks is still Gradient Descent (GD). To apply GD, we need to calculate the gradient of the loss function with respect to each weight matrix $W^{(l)}$ and bias vector $b^{(l)}$. First, we need to compute the output predicted by the MLP network, denoted as $\hat{y}$ corresponding to an input value x.

$$a^{(0)} = x$$

Then, the input vector of the l^{th} layer is:

$$z^{(l)} = W^{(l)} a^{(l-1)} + b^{(l)}, \quad l = 1, 2, ..., L$$

and its output vector is:

$$a^{(l)} = f(z^{(l)}), \quad l = 1, 2, ..., L$$

Finally, the predicted value is also the output of the last layer:

$$\hat{y} = a^{(L)}$$

This step is called feedforward because the computation is performed from the beginning to the end of the MLP network.

Assume $J(W, b, X, Y)$ is a loss function of the problem, where W and b are sets of all weight matrices and biases in the network, and X and Y are the training data pairs with each column corresponding to a data point. To apply gradient-based methods (of which Gradient Descent is an example), we need to compute:

$$\frac{\partial J}{\partial W^{(l)}}; \frac{\partial J}{\partial b^{(l)}}, \quad l = 1, 2, ..., L$$

A common loss function is the Mean Square Error (MSE) defined as:

$$J(W, b, X, Y) = \frac{1}{N} \sum_{n=1}^{N} ||y_n - \hat{y}_n||_2^2$$

With N being the number of data pairs (x, y) in the training set.

According to the formulas above, direct computation of these values is highly complex because the loss function does not directly depend on these variables. The most common method used is called Backpropagation, which computes the gradient backward from the last layer to the first layer. The last layer is computed first because it involves the predicted value and the loss function. Computing the gradients of previous layers is based on a specific rule called the chain rule, which means the derivative of a composite function.

As such, the derivative of the loss function with respect to the weights of the last layer is calculated as follows:

$$\frac{\partial J}{\partial w_{ij}^{(L)}} = \frac{\partial J}{\partial z_j^{(L)}} \cdot \frac{\partial z_j^{(L)}}{\partial w_{ij}^{(L)}} = e_j^{(L)} \cdot a_i^{(L-1)}$$

where:

- $e_j^{(L)} = \dfrac{\partial J}{\partial z_j^{(L)}} = \dfrac{\partial J}{\partial a_j^{(L)}} \cdot \dfrac{\partial a_j^{(L)}}{\partial z_j^{(L)}} = \left(\sum_{k=1}^{d(L+1)} \dfrac{\partial J}{\partial z_k^{(L+1)}} \cdot \dfrac{\partial z_k^{(L+1)}}{\partial a_j^{(L)}} \right) \cdot f'(z_j^{(L)})$
- $f'(z_j^{(L)}) = \dfrac{df}{dz}(z_j^{(L)})$

Similarly, we can calculate the derivative of the loss function with respect to the bias of the last layer as:

$$\frac{\partial J}{\partial b_j^{(L)}} = e_j^{(L)}$$

From the formulas above, we can calculate the derivatives of weights at previous layers as follows:

$$\frac{\partial J}{\partial w_{ij}^{(l)}} = \frac{\partial J}{\partial z_j^{(l)}} \cdot \frac{\partial z_j^{(l)}}{\partial w_{ij}^{(l)}} = e_j^{(l)} \cdot a_i^{(l-1)}$$

where:

$$e_j^{(l)} = \frac{\partial J}{\partial z_j^{(l)}} = \sum_{k=1}^{d(l+1)} e_k^{(l+1)} \cdot w_{jk}^{(l+1)} \cdot f'(z_j^{(l)})$$

Similarly, we can compute the derivative of the loss function with respect to the bias of previous layers as:

$$\frac{\partial J}{\partial b_j^{(l)}} = e_j^{(l)}$$

Algorithm Overview

We denote:

- L as the loss function.
- **w** as the vector of weights in the network.
- $\frac{\partial L}{\partial \mathbf{w}}$ as the gradient of the loss function with respect to the weights.
- **x** as the input to the network.
- **y** as the target output.
- **z** as the output of the network (before applying the activation function).

The backpropagation algorithm can be summarized as follows:

1. **Forward Pass:** Compute the output of the network **z** given the input **x**.
2. **Compute Loss:** Calculate the loss L between the predicted output **z** and the target output **y**.
3. **Backward Pass:** Compute the gradients of the loss function with respect to the weights $\frac{\partial L}{\partial \mathbf{w}}$ using the chain rule.

4. **Update Weights:** Update the weights $\mathbf{w}$ using an optimization algorithm such as gradient descent:

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \frac{\partial L}{\partial \mathbf{w}}$$

where α is the learning rate.

```
function Backpropagation(input_data, target_output):
    // Forward Pass
    output = NeuralNetwork.forward(input_data)
    // Compute Loss
    loss = LossFunction.compute(output, target_output)
    // Backward Pass
    gradients = LossFunction.backward(output, target_output)
    NeuralNetwork.backward(gradients)
    // Update Weights
    NeuralNetwork.update_weights(learning_rate)
```

3.3 Loss Functions

In the realm of machine learning, a loss function, also referred to as a cost function or objective function, serves as a pivotal metric for quantifying the disparity between a model's predictions and the actual ground truth. The fundamental objective during the training phase of a machine learning model is to minimize this loss function, which essentially guides the iterative process of adjusting the model's parameters to enhance predictive accuracy.

A loss function must satisfy 3 factors:

- **Differentiability**: Loss functions must possess differentiability with respect to the model parameters, facilitating the utilization of gradient-based optimization algorithms like SGD.
- **Interpretability**: Loss functions should offer interpretable feedback on the model's performance, enabling stakeholders to comprehend and assess the training progress comprehensively.

- **Robustness**: To ensure stable training and mitigate overfitting, loss functions should demonstrate robustness in the presence of outliers and noise within the dataset.

We will introduce some commonly used loss functions as follow:

3.3.1 Mean Squared Error

Mean Squared Error (MSE) is a common choice for regression tasks. It computes the average squared discrepancy between the model's predictions and the true target values:

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

where:

- N: Number of samples
- y_i: True target value for sample i
- $\hat{y}_i$: Predicted target value for sample i

3.3.2 Binary Cross-Entropy Loss

Binary Cross-Entropy (BCE) Loss is commonly used in binary classification tasks. It measures the disparity between predicted probabilities of the positive class and the corresponding true labels, employing logarithmic transformations to penalize misclassifications significantly:

$$\text{BCE} = -\frac{1}{N}\sum_{i=1}^{N}[y_i \cdot \log(\hat{y}_i) + (1 - y_i) \cdot \log(1 - \hat{y}_i)]$$

where:

- N: Number of samples
- y_i: True target value for sample i
- $\hat{y}_i$: Predicted target value for sample i

3.3.3 Categorical Cross-Entropy Loss

Categorical Cross-Entropy (CCE) Loss is tailored for multiclass classification undertakings. It extends the principles of binary cross-entropy loss to handle multiple classes, computing the average negative log likelihood of the true class labels:

$$\text{CCE} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{C} y_{ij} \cdot \log(\hat{y}_{ij})$$

where:

- C: Number of classes
- y_{ij}: True label for sample i and class j
- $\hat{y}_{ij}$: Predicted probability for sample i and class j

3.3.4 Hinge Loss

Hinge Loss is often used in binary classification scenarios with Support Vector Machines (SVMs). It penalizes misclassifications linearly, thereby facilitating the maximization of margins between classes:

$$\text{Hinge} = \frac{1}{N}\sum_{i=1}^{N} \max(0, 1 - y_i \cdot \hat{y}_i)$$

where:

- N: Number of samples
- y_i: True target value for sample i
- $\hat{y}_i$: Predicted target value for sample i

3.3.5 Kullback-Leibler Divergence

Kullback-Leibler Divergence (KL Divergence) is a metric employed to measure the dissimilarity between two probability distributions. It finds prominent application in generative models such as VAEs and GANs to quantify the distinction between the model's

distribution and the genuine data distribution:

$$\mathrm{KL}(P||Q) = \sum_{x \in X} P(x) \cdot \log\left(\frac{P(x)}{Q(x)}\right)$$

where:

- $\mathrm{KL}(P||Q)$: Kullback-Leibler Divergence from distribution Q to distribution P
- $P(x)$: Probability of observation x in distribution P
- $Q(x)$: Probability of observation x in distribution Q
- X: Support set of the distributions

This formulation measures how much information is lost when the distribution Q is used to approximate the true distribution P. A value of zero indicates perfect similarity between the distributions, while higher values indicate greater dissimilarity.

3.4 Convolutional Neural Network

Convolutional Neural Networks (CNNs) are a class of deep neural networks that have revolutionized the field of computer vision. They are specifically designed to efficiently and automatically learn hierarchical representations of visual data. CNNs have achieved remarkable success in various tasks such as image classification, object detection, facial recognition, and image segmentation.

Components

CNNs consist of several layers, each serving a specific purpose in the feature extraction process. The key components of a typical CNN architecture include:

1. **Convolutional Layers:** These layers perform convolution operations to extract relevant features from the input data. Convolutional filters learn to detect patterns such as edges, textures, and shapes at different spatial locations in the input images.
2. **Pooling Layers:** Pooling layers reduce the spatial dimensions of the feature maps produced by the convolutional layers. They aggregate information within local

regions to capture the most important features while reducing computational complexity and preventing overfitting.

3. **Activation Functions:** Activation functions introduce non-linearity into the network, allowing it to learn complex mappings between input and output. Common activation functions used in CNNs include ReLU, sigmoid, and tanh.

4. **Fully Connected Layers:** Fully connected layers are traditional neural network layers where every neuron is connected to every neuron in the previous and subsequent layers. They perform classification based on the high-level features extracted by the convolutional layers.

5. **Normalization Layers:** Normalization layers are used to normalize the activations of the network, helping stabilize and accelerate the training process.

6. **Dropout Layers:** Dropout layers are a regularization technique used to prevent overfitting by randomly dropping a fraction of neurons during training.

7. **Output Layer:** The output layer produces the final predictions of the network, typically using a softmax activation function for multi-class classification tasks or a linear activation function for regression tasks.

3.4.1 Convolution

The convolution operation involves sliding a small filter (kernel) over the input image and computing element-wise multiplications followed by summation to produce the output feature map. The key components of the convolution operation are:

1. **Kernel:** A small matrix of learnable parameters that extracts specific features from the input image.

2. **Stride:** The step size by which the kernel moves horizontally and vertically across the input image.

3. **Padding:** Adding extra border pixels to the input image to control the spatial dimensions of the output feature map.

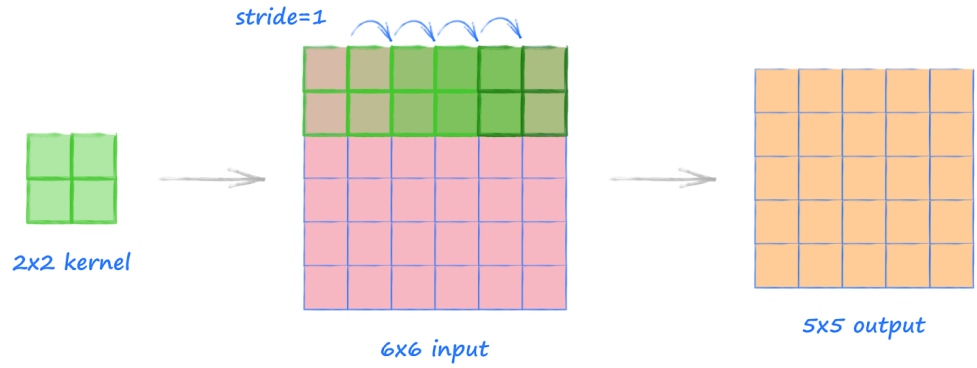


Figure 3.5: Filter and stride in CNN

The convolution operation can be mathematically expressed as:

$$\mathbf{O}(i,j) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} \mathbf{I}(i+m, j+n) \cdot \mathbf{K}(m,n)$$

where:

- $\mathbf{O}(i,j)$: Output feature map value at position (i,j).
- $\mathbf{I}(i,j)$: Input image value at position (i,j).
- $\mathbf{K}(m,n)$: Kernel value at position (m,n).
- M and N: Dimensions of the kernel.

3.4.2 Pooling

Pooling is a downsampling operation commonly applied after convolutional layers in CNNs. It aims to reduce the spatial dimensions of the input while retaining the most important information. The two most common types of pooling are max pooling and average pooling.

Max Pooling

In max pooling, for each local region of the input feature map, the maximum value is taken. This is achieved by sliding a window (often referred to as a pooling kernel or filter) over the input and selecting the maximum value within each window. The size of

the window and the stride (the amount by which the window shifts) are hyperparameters that determine the degree of downsampling.

Given an input feature map **X** of size $W \times H \times D$, where W is the width, H is the height, and D is the depth (number of channels), max pooling with a window size of $F \times F$ and a stride of S produces an output feature map **Y** of size:

$$\frac{W-F}{S}+1 \times \frac{H-F}{S}+1 \times D$$

Average Pooling

In average pooling, instead of taking the maximum value, the average value within each window is computed. Like max pooling, the window size and stride are hyperparameters that determine the degree of downsampling.

Average pooling operates similarly to max pooling, but instead of selecting the maximum value within each window, it computes the average value.

Just like max pooling, average pooling with a window size of $F \times F$ and a stride of S produces an output feature map **Y** of size:

$$\frac{W-F}{S}+1 \times \frac{H-F}{S}+1 \times D$$

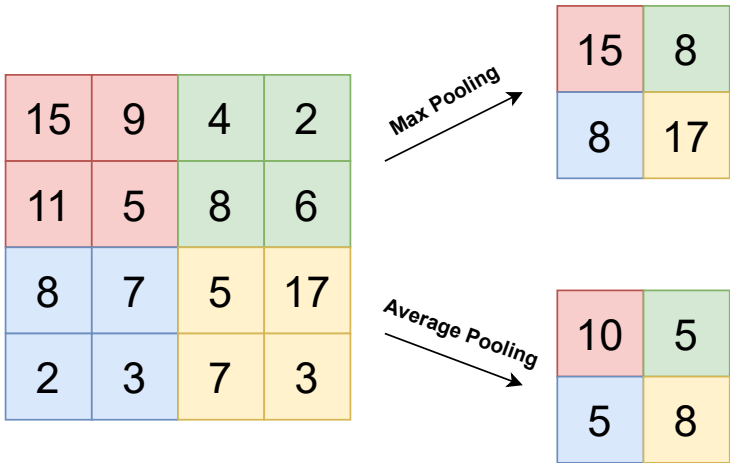


Figure 3.6: Pooling demonstrations with Max Pooling and Average Pooling

3.5 Recurrent Neural Network

Recurrent Neural Networks (RNNs) are a class of artificial neural networks designed to handle sequential data by maintaining a form of memory. Unlike feedforward neural

networks, which process input data in a single direction, from input layer to output layer, RNNs have connections that form directed cycles, allowing them to exhibit temporal dynamic behavior. This cyclic structure enables RNNs to effectively model sequences of data, making them suitable for tasks involving sequential dependencies.

RNNs are characterized by their ability to maintain a hidden state that captures information about previous inputs seen by the network. This hidden state serves as a form of memory, allowing RNNs to incorporate context from past inputs when making predictions or decisions about future outputs. The recurrent nature of RNNs makes them well-suited for tasks such as time series prediction, language modeling, machine translation, and sentiment analysis.

In the field of NLP, RNNs have emerged as a powerful tool for modeling sequential data inherent in language. One of the key advantages of RNNs in NLP is their ability to capture long-range dependencies in text data, which is crucial for tasks such as language understanding, generation, and translation. RNNs are commonly used in NLP tasks such as language modeling, where the goal is to predict the next word in a sequence given the previous words. By learning the underlying structure of language through training on large text corpora, RNN-based language models can generate coherent and contextually relevant text. Another important application of RNNs in NLP is machine translation, where the goal is to translate text from one language to another. RNNs, particularly in the form of sequence-to-sequence models, have demonstrated state-of-the-art performance in this task by effectively capturing the semantic and syntactic nuances of language.

Furthermore, RNNs are used in sentiment analysis, named entity recognition, text summarization, and various other NLP tasks where sequential data processing is required.

3.5.1 Forward Propagation

The forward pass in RNNs involves processing sequential data through the network to generate predictions or extract features.

We denote:

- $x^{(t)}$ represents the input at time step t.
- $h^{(t)}$ denotes the hidden state at time step t.

- $y^{(t)}$ is the output at time step t.
- W_{hx}, W_{hh}, and W_{yh} represent the weight matrices for input-to-hidden, hidden-to-hidden, and hidden-to-output connections, respectively.
- b_h and b_y denote the bias vectors for the hidden and output layers, respectively.

In a simple RNN architecture, the forward pass can be described as follows:

1. **Input Processing**: At each time step t, the input $x^{(t)}$ is processed using the input-to-hidden weight matrix W_{hx} and the hidden-to-hidden weight matrix W_{hh}, along with the bias b_h. This is expressed by the following equations:

$$h^{(t)} = \tanh(W_{hx}x^{(t)} + W_{hh}h^{(t-1)} + b_h)$$

2. **Output Generation**: The hidden state $h^{(t)}$ is then used to generate the output $y^{(t)}$ using the hidden-to-output weight matrix W_{yh} and the bias b_y:

$$y^{(t)} = \text{softmax}(W_{yh}h^{(t)} + b_y)$$

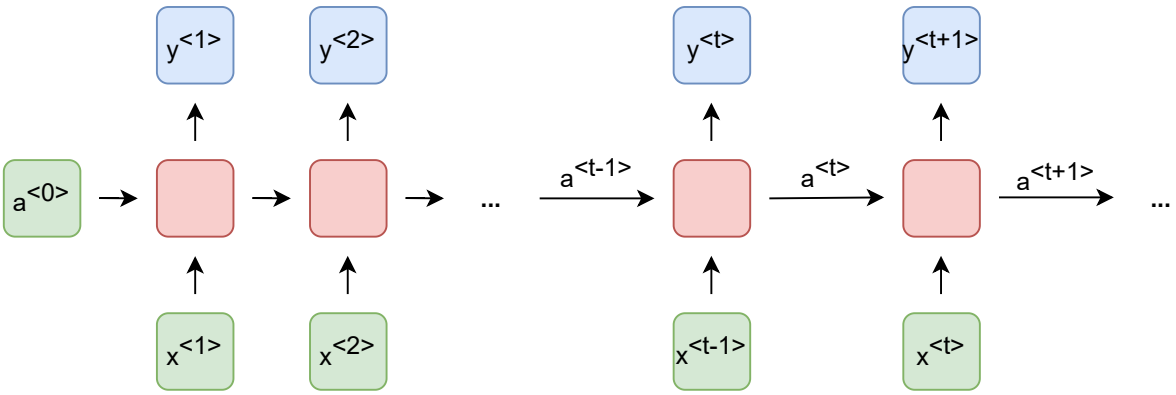


Figure 3.7: RNN General Architecture

In general, we have the following RNN architectures:

- **One-to-One**:

  The One-to-One forward pass refers to a straightforward feedforward pass similar to traditional neural networks. In this type of forward pass, each input corresponds to a single output without any temporal dependencies. It's essentially a

non-recurrent operation, and the RNN architecture behaves like a standard feed-forward neural network.

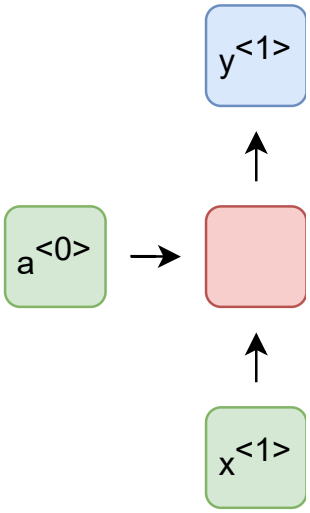


Figure 3.8: One-to-One RNN architecture

- **One-to-Many**:

  In a One-to-Many forward pass, a single input is mapped to a sequence of outputs. This type of forward pass is useful for tasks such as image captioning, where a single image is used to generate a sequence of words describing the contents of the image. The input is fed into the RNN only once, and the network generates multiple outputs over time.

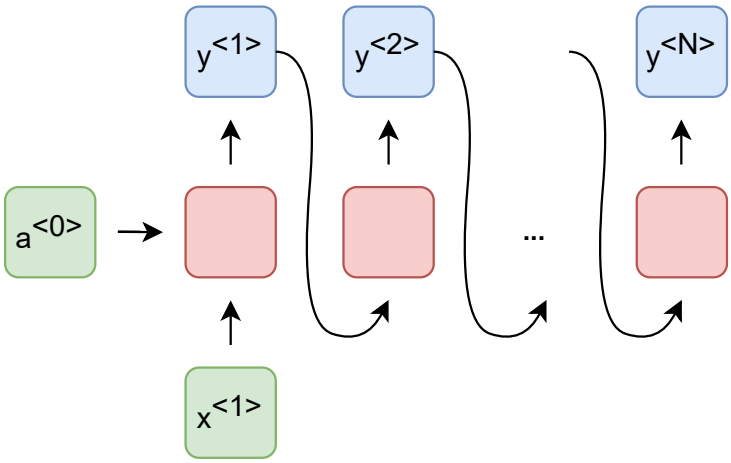


Figure 3.9: One-to-Many RNN architecture

- **Many-to-One**:

In a Many-to-One forward pass, a sequence of inputs is processed by the RNN, and a single output is produced at the end of the sequence. This type of forward pass is commonly used for tasks such as sentiment analysis, where the sentiment of a text is determined based on the entire sequence of words.

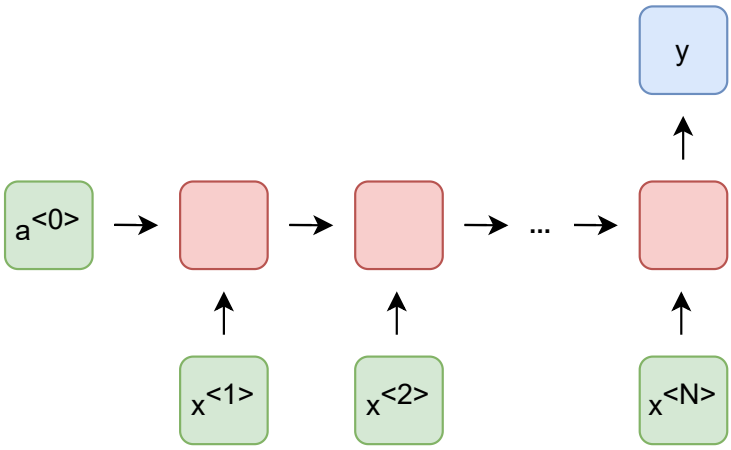


Figure 3.10: Many-to-One RNN architecture

- **Many-to-Many**:

  The Many-to-Many forward pass involves both input and output sequences of variable lengths. This type of forward pass is suitable for tasks such as sequence labeling, where each input is associated with a sequence of labels, and each label is generated based on the corresponding input.

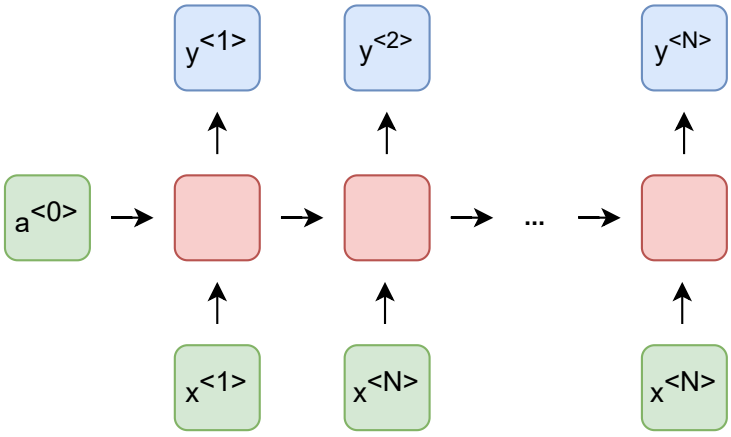


Figure 3.11: Many-to-Many RNN architecture

3.5.2 Loss Function

RNNs are often employed in sequence prediction tasks such as language modeling, machine translation, and time series forecasting. In such tasks, the network outputs a sequence of predictions, and a loss function is used to measure the discrepancy between these predictions and the ground truth sequence.

In the case of RNNs, the loss function L for all steps is determined based on the loss at each step. The loss function at each step is defined as:

$$L(\hat{y}^{<t>}, y^{<t>}) = \sum_{t=1}^{T} L(\hat{y}^{<t>}, y^{<t>}) \tag{3.2}$$

where:

- $\hat{y}^{<t>}$ represents the model's predicted value at step t.
- $y^{<t>}$ represents the actual value at step t.
- T_y is the length of the output sequence. Similarly, T_y is the length of the input sequence. In the case of many-to-one architecture, $T_y = 1$.

One commonly used loss function for RNNs is the *cross-entropy loss*, especially in tasks involving classification. Given a sequence of predicted probability distributions $\hat{y}_1, \hat{y}_2, ..., \hat{y}_T$ and the corresponding true distributions $y_1, y_2, ..., y_T$, the cross-entropy loss is calculated as:

$$L = -\frac{1}{T} \sum_{t=1}^{T} \sum_{i=1}^{N} y_{t,i} \log(\hat{y}_{t,i}) \tag{3.3}$$

where T is the length of the sequence, N is the number of classes, $y_{t,i}$ represents the true probability of class i at time step t, and $\hat{y}_{t,i}$ is the predicted probability of class i at time step t.

In addition to cross-entropy loss, other loss functions such as MSE can also be used depending on the nature of the task. MSE is typically employed in regression tasks where the network predicts continuous values.

3.5.3 Back-propagation through Time

The Back-Propagation Through Time (BPTT) algorithm is a variant of the standard back-propagation algorithm used for training RNNs. In BPTT, the network is unfolded through

time, creating a computational graph where each timestep corresponds to a layer in the unfolded network. The forward pass involves computing the output sequence given an input sequence, while the backward pass involves computing gradients with respect to the parameters of the network.
The steps of the BPTT algorithm can be summarized as follows:

1. **Forward Pass**: Perform a forward pass through the unfolded network to compute the predicted output sequence.

2. **Loss Computation**: Compute the loss between the predicted output sequence and the ground truth sequence using a chosen loss function.

3. **Backward Pass**: Perform a backward pass through the unfolded network to compute gradients of the loss with respect to the parameters of the network.

4. **Gradient Descent**: Update the parameters of the network using gradient descent or its variants, such as Adam or RMSProp, to minimize the loss.

During the backward pass, the gradients are accumulated across time steps using the chain rule of calculus, similar to how it is done in standard back-propagation. However, in RNNs, the gradients can potentially explode or vanish as they are back-propagated through many time steps. This issue is commonly referred to as the vanishing gradient problem.
To mitigate the vanishing gradient problem, techniques such as gradient clipping, using non-saturating activation functions like ReLU, and employing architectures like Long Short-Term Memory or Gated Recurrent Unit have been developed.

3.6 Long-Short Term Memory

3.6.1 Foundation

Long Short-Term Memory (LSTM) is a type of RNN architecture designed to overcome the limitations of traditional RNNs in capturing long-term dependencies in sequential data. Introduced by Hochreiter and Schmidhuber in 1997, LSTM networks have become a cornerstone in the field of deep learning, particularly in tasks involving sequential data

like speech recognition, language modeling, machine translation, and time series prediction.

Basically, LSTM networks address the vanishing gradient problem that plagues traditional RNNs. This problem arises when training RNNs on long sequences, causing gradients to diminish exponentially as they propagate through time, making it challenging for the network to learn from distant past information. LSTM achieves this by incorporating specialized units called memory cells, which allow the network to selectively remember or forget information over time.

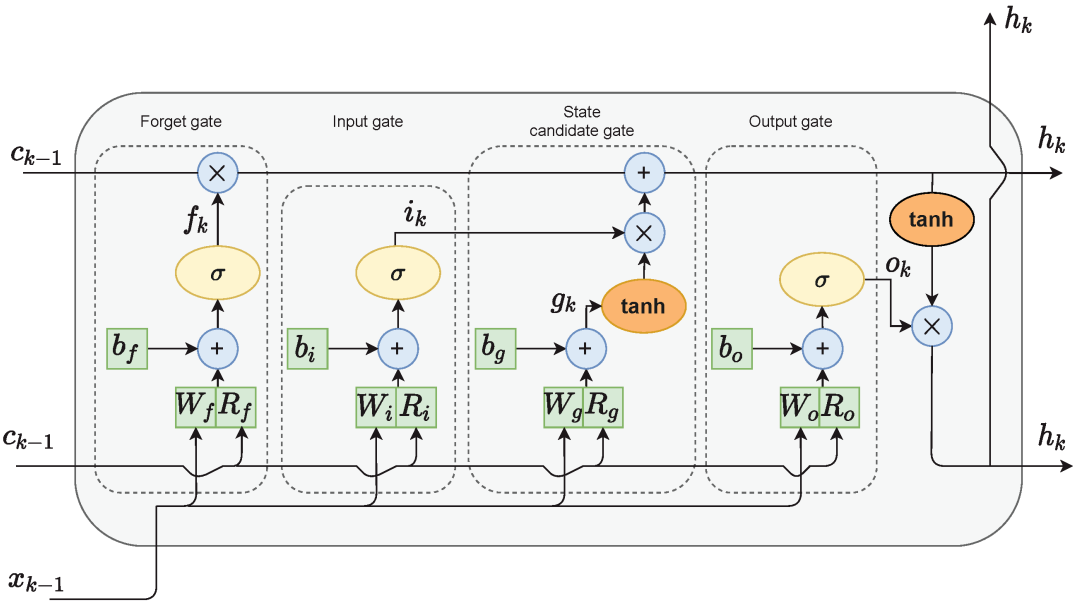


Figure 3.12: LSTM architecture and its cell structure[1]

Each LSTM unit consists of three main components: the input gate, the forget gate, and the output gate, along with a memory cell. These gates are responsible for controlling the flow of information into and out of the memory cell, enabling the network to retain relevant information and discard irrelevant details.

3.6.2 Notation

We denote:

- x_t: input at time step t,

[1] https://www.mdpi.com/1424-8220/21/16/5625

- h_t: hidden state at time step t,
- C_t: cell state at time step t,
- f_t: forget gate activation at time step t,
- i_t: input gate activation at time step t,
- o_t: output gate activation at time step t,
- W and U: weight matrices,
- b: bias vector.

The LSTM equations can be represented as follows:

$$\begin{aligned}
\Gamma_i &= \sigma(W_i x_t + W_{hi} h_{t-1} + b_i) \\
\Gamma_f &= \sigma(W_f x_t + W_{hf} h_{t-1} + b_f) \\
\Gamma_o &= \sigma(W_o x_t + W_{ho} h_{t-1} + b_o) \\
\tilde{c}_t &= \text{tanh}(W_c x_t + W_{hc} h_{t-1} + b_c) \\
c_t &= \Gamma_f \odot c_{t-1} + \Gamma_i \odot \tilde{c}_t \\
h_t &= \Gamma_o \odot \text{tanh}(c_t)
\end{aligned}$$

where:

- x_t is the input at time step t.
- $\tilde{y}_t$ is the output at time step t.
- Γ_i is the input gate.
- Γ_f is the forget gate.
- Γ_o is the output gate.
- c_t is the internal state (meaning the cell state or memory) at time step t.
- $\tilde{c}_t$ represents the candidate state for the internal state at time step t.
- W_i, W_{hi}, W_f, W_{hf}, W_o, W_{ho}, W_c, W_{hc}, b_i, b_f, b_o, b_c are the weights and biases of the gates and memory cell.

- $\odot$ denotes element-wise multiplication between matrices.

The gates in LSTM can have real values in the range from 0 to 1 due to the nature of the sigmoid function σ. If the value is 0, then all information will be discarded, whereas if the value is 1, then all information will be retained. The output gate indicates which information will be passed to the next hidden state.

During the forward pass, the input gate regulates the flow of new information into the memory cell, the forget gate decides which information to discard from the cell's memory, and the output gate determines the information to be outputted to the next timestep or layer. The memory cell itself serves as a kind of conveyor belt that runs through time, allowing information to persist over long sequences.

3.7 Tokenizer

In LLMs, tokenization plays a critical role in preprocessing text data before feeding it into the model. Tokenization in LLMs involves converting raw text into a sequence of tokens, where each token typically represents a word or a subword. However, unlike traditional tokenization methods, LLMs often employ more sophisticated techniques to handle the complexities of natural language.

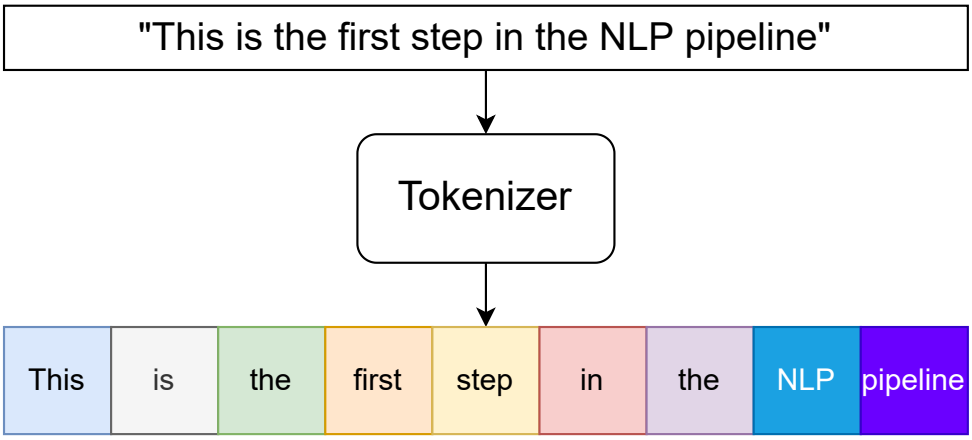


Figure 3.13: An example of Tokenizer

Tokenization in LLMs include:

- **Subword Tokenization**: This approach breaks words into smaller subword units, which helps in dealing with rare words, misspellings, and morphological varia-

tions. Subword tokenization algorithms like Byte Pair Encoding [39] and WordPiece [41] are commonly used.

- **Special Tokens**: LLMs often include special tokens to denote various linguistic features or special instructions. For example, `[CLS]` token is added at the beginning of input sequences for classification tasks, `[SEP]` token separates segments of text, and `[MASK]` token is used during pre-training for masked language modeling objectives.

- **Vocabulary Size**: The vocabulary size in LLMs is typically large, covering a wide range of words and subwords encountered in the training data. However, to manage memory and computational resources, most LLMs use a fixed-size vocabulary and handle Out-Of-Vocabulary (OOV) words through subword tokenization.

- **Normalization**: Text normalization techniques may be applied during tokenization to standardize the text and improve model performance. This may include lowercasing, removing accents, punctuation, or special characters, and handling numerical expressions.

3.8 Transformer

The Transformer architecture represents a pivotal advancement in the field of NLP, revolutionizing various tasks such as machine translation, text summarization, and language understanding. Introduced by Vaswani et al. in 2017, the Transformer model eliminates the sequential nature of RNNs and the fixed-length limitations of CNNs, offering a parallelizable and scalable alternative for processing sequential data.

At the heart of the Transformer lies the self-attention mechanism, which enables the model to weigh the importance of different words in a sentence when encoding or decoding. This attention mechanism allows the Transformer to capture long-range dependencies within sequences, facilitating more effective information processing compared to traditional sequential models.

The Transformer architecture consists of encoder and decoder components, each comprising multiple layers of self-attention mechanisms and feed-forward neural networks.

In the encoder, self-attention layers analyze the input sequence in parallel, capturing contextual information from all tokens simultaneously. Similarly, the decoder utilizes self-attention layers to attend to different parts of the input sequence while generating an output sequence, enabling the model to maintain coherence and capture dependencies during both encoding and decoding phases.

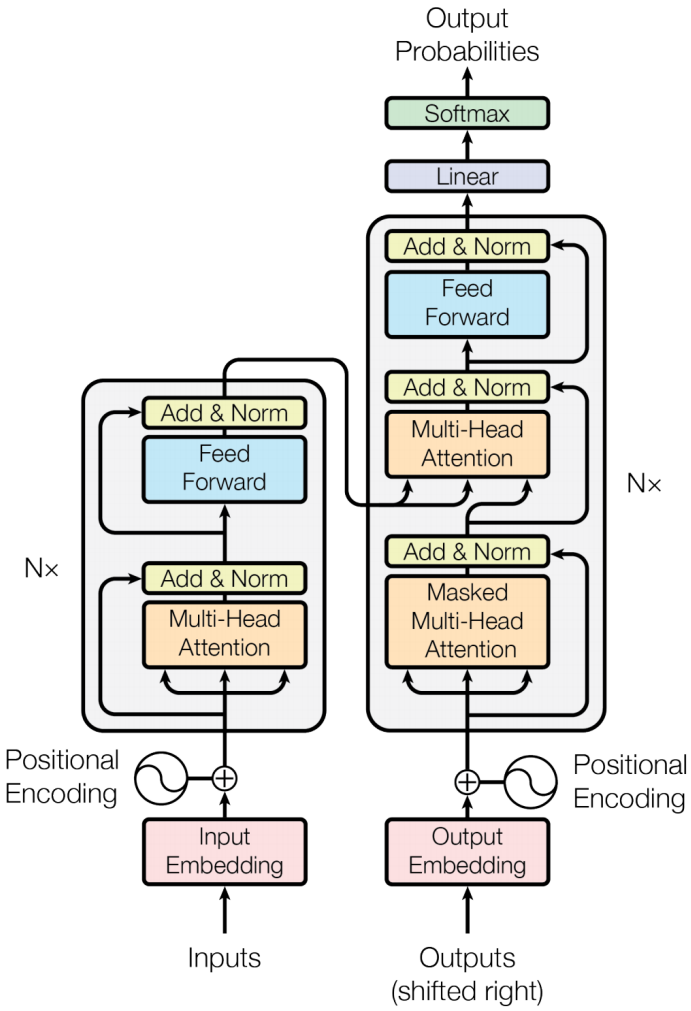


Figure 3.14: Transformer architecture from paper ”Attention Is All You Need” [49]

Input Embedding. In the Transformer, input embeddings are typically learned as part of the model training process, where each token in the input sequence is mapped to a unique embedding vector. These embedding vectors are then combined with positional encodings to incorporate information about the token’s position in the sequence. The input embeddings serve as the input to the self-attention mechanism, allowing the model to capture contextual information and dependencies between tokens. By learning mean-

ingful representations of input tokens, the Transformer can effectively process sequential data and perform various natural language processing tasks with high accuracy.

Positional Encoding. Since Transformers lack recurrence mechanisms like those found in RNNs, positional encodings are added to the input embeddings to provide information about the position of each token in the sequence. To incorporate positional information into the input embeddings, researchers proposed using a combination of sine and cosine functions to create positional vectors. This approach allows the Transformer to handle sequences of varying lengths by providing a flexible method for encoding positional information. In this approach, each dimension of the positional encoding is represented by unique frequencies and offsets of sine and cosine waves. These frequencies and offsets enable the model to capture positional information in a continuous and differentiable manner, ensuring that the positional encoding can be effectively learned during the training process. The values of the positional encoding range from -1 to 1, effectively representing each position within the sequence. By adding these positional encodings to the input embeddings, the Transformer model gains the ability to understand and leverage the sequential order of tokens, thereby improving its performance on tasks such as natural language understanding and generation.

Scaled Dot-Product. Each word in the input sequence is associated with three vectors: A Query Q, a Key K, and a Value V. These vectors are used to compute attention scores, which determine the importance of each word in relation to the others. When a query and a key have a high attention score, the corresponding value is emphasized in the output of the attention layer. The similarity is measured by dot product as:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

where d_k is the dimensionality of the key vectors.

Multi-head Attention. In the multi-head attention mechanism, the input sequence is linearly projected h times into query, key, and value spaces, where h represents the number of attention heads. Attention is then computed in parallel for each of these projected versions, resulting in h-dimensional output values.

$$\text{MultiHead}(X) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O$$

where each head_i is a single attention head, and W_O is a learnable weight matrix used

for the output projection.

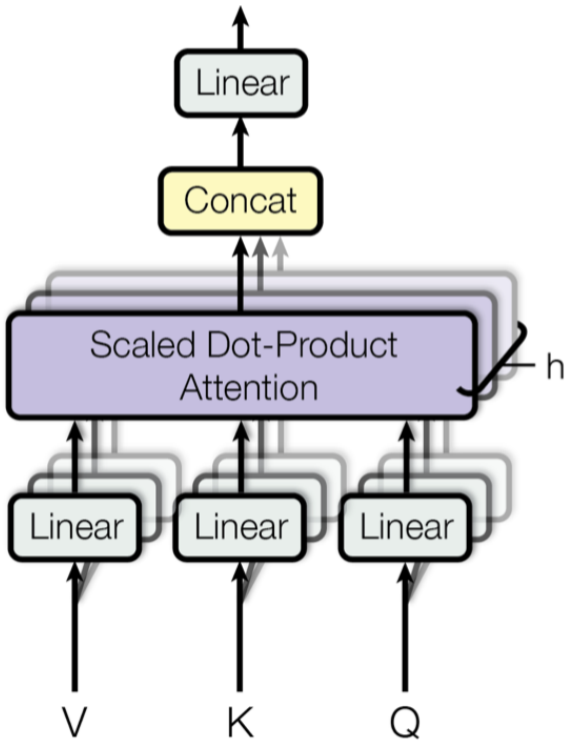


Figure 3.15: Multi-head Attention architecture from paper ”Attention Is All You Need” [49]

Normalization and Residual Connections. Normalization techniques, such as layer normalization or batch normalization, are applied after each layer in the network to stabilize the training process and improve the convergence of the model. These techniques help address issues like vanishing or exploding gradients by ensuring that the inputs to each layer are normalized and have similar scales. This allows the model to learn more efficiently and reduces the risk of optimization difficulties during training. Residual connections, also known as skip connections, involve adding the input of a layer to its output before passing through a nonlinear activation function. This technique allows gradients to flow more easily through the network during training, mitigating the vanishing gradient problem and enabling the model to learn deeper representations. Residual connections also help prevent the degradation of performance that can occur in very deep networks by providing shortcuts for information to bypass certain layers.

Feed-Forward Neural Network. Feed-forward Neural Networks (FFNNs) are applied after the self-attention mechanism in both the encoder and decoder layers. These FFNNs

consist of two fully connected layers with a nonlinear activation function (typically ReLU) applied between them. The FFNNs enable the model to learn complex patterns and relationships within the input data by performing nonlinear transformations on the output of the self-attention mechanism.

Masked Self-Attention Mechanism. It operates similarly to the self-attention mechanism in the encoder but with a crucial difference: it prevents positions from attending to subsequent positions in the sequence. This restriction ensures that each word in the sequence is influenced only by previous tokens and not by future ones. For example, when computing attention scores for a word like ”are,” the masked self-attention mechanism ensures that ”are” cannot attend to subsequent words like ”you” in the sequence. This prevents the model from peeking ahead during decoding and ensures that predictions for a particular position are based only on known outputs at positions before it.

3.9 Multimodal Models

Multimodal models are a class of artificial intelligence models that are designed to process and integrate information from multiple modalities, such as text, images, audio, or other types of data. These models have gained increasing attention due to their ability to capture rich and diverse information from different sources, leading to improved performance on various tasks compared to unimodal models. The key characteristic of multimodal models is their capability to understand and reason about data from different modalities simultaneously. By combining information from multiple modalities, these models can leverage complementary cues and contextual information to make more accurate predictions or generate more informative outputs.

Early Fusion. In early fusion, the data from different modalities are combined at the input level before being processed by the model. For example, in a multimodal classification task, features extracted from images and text can be concatenated or merged before being fed into a neural network for classification.

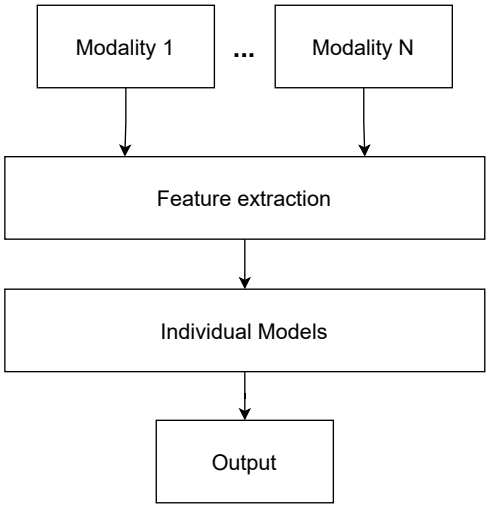


Figure 3.16: Early Fusion pipeline

Late Fusion. In contrast to early fusion, late fusion involves processing data from each modality separately and then combining the outputs of individual models at a later stage. This approach allows for more flexibility in modeling each modality independently before integrating the information.

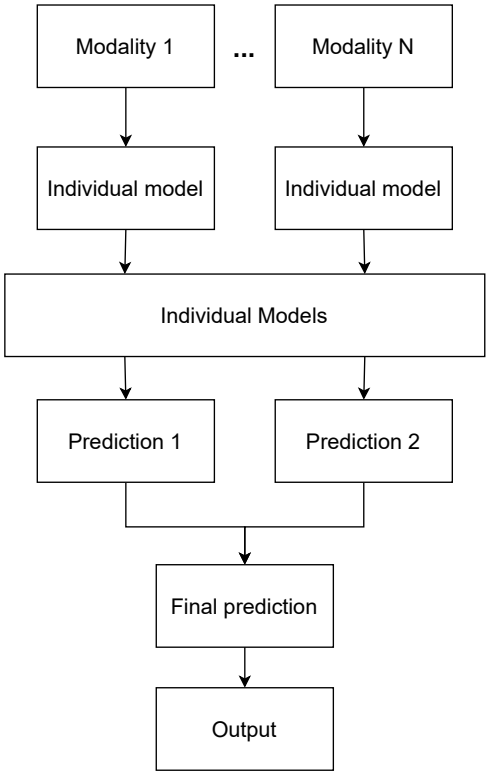


Figure 3.17: Late Fusion pipeline

Cross-Modal Attention. Cross-modal attention mechanisms enable the model to dy-

namically attend to relevant information from different modalities during processing. These mechanisms allow the model to selectively focus on relevant features from each modality based on the context of the task.

Transformer-based approaches have extended their capabilities to handle multimodal inputs effectively. Models like ViT and CLIP leverage the Transformer architecture to process image data directly and learn joint representations of images and text, enabling tasks such as image classification, object detection, image-text retrieval, and zero-shot classification. BERT-based multimodal models, such as UNITER [7], combine BERT with vision transformers to capture interactions between text and image modalities, facilitating tasks like visual question answering and image-text retrieval. Graph-based multimodal models, like MM-GNN [12], represent different modalities as nodes in a graph and use graph neural networks to capture complex relationships between them, enabling tasks such as recommendation systems and knowledge graph completion. Attention-based multimodal models, exemplified by MMBT [20] and LXMERT [42], utilize attention mechanisms to selectively attend to relevant information from different modalities, achieving state-of-the-art performance on tasks like image captioning and visual question answering.

3.10 Large Language Models

Large language models represent a significant advancement in NLP research, leveraging deep learning architectures to comprehend and generate human-like text at scale. These models are trained on vast amounts of text data, learning complex patterns and relationships to perform a wide range of language understanding tasks. At the core of large language models lie architectures such as Transformers, which have revolutionized NLP by enabling efficient processing of long-range dependencies in text. The Transformer architecture relies on self-attention mechanisms to capture contextual information from input sequences, enabling models to understand and generate coherent text across various domains and languages.

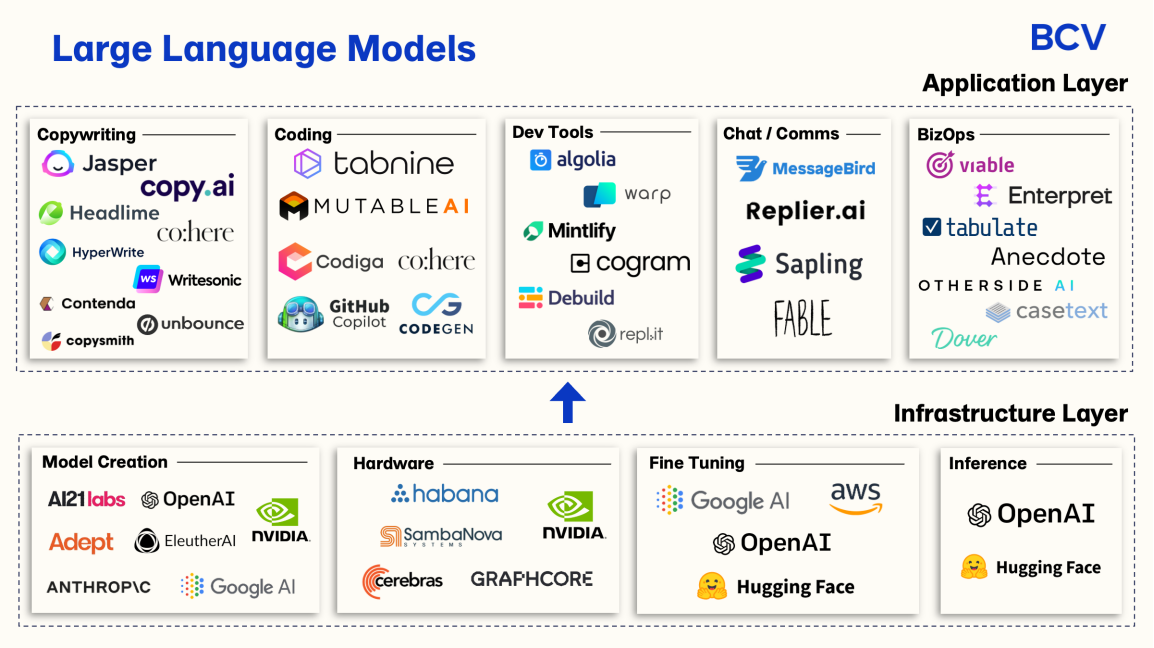


Figure 3.18: Application of LLMs[1]

In recent years, several large language models have gained prominence for their impressive performance on benchmark tasks and their ability to generate high-quality text. Among the most notable models is OpenAI's GPT (Generative Pre-trained Transformer) series, which includes models such as GPT-2 and GPT-3. These models are pre-trained on massive datasets and fine-tuned for specific downstream tasks, achieving state-of-the-art results in areas like language translation, text summarization, and question answering. GPT-3, in particular, with its unprecedented scale of 175 billion parameters, has demonstrated remarkable capabilities in natural language understanding and generation, showcasing the potential of large language models to advance the field of NLP.

In addition to GPT, other large language models have emerged from leading research institutions and tech companies. BERT model introduced a pre-training technique based on bidirectional transformers, significantly improving performance on various NLP tasks. Models like RoBERTa [30], XLNet [53], and T5 (Text-To-Text Transfer Transformer) [37] have further refined pre-training methodologies and achieved state-of-the-art results across a wide range of tasks, including text classification, language modeling, and machine translation.

One prominent category of LLMs is the Encoder-only LLMs, which rely solely on the

[1] https://baincapitalventures.com/insight/large-language-models-will-redefine-b2b-software/

encoder module to process input sentences and discern the relationships between words. These models are typically pre-trained on large-scale corpora using tasks like masked language modeling. To adapt to downstream tasks such as text classification and named entity recognition, additional prediction heads are appended to the encoder module. Notable examples include BERT, RoBERTa, ALBERT [22], and ELECTRA [8], which excel at tasks requiring comprehensive understanding of entire sentences.

In contrast, Encoder-decoder LLMs encompass both encoder and decoder modules, with the encoder responsible for encoding input sentences into a latent space and the decoder utilized for generating target output text. This architecture offers greater flexibility in training strategies and is particularly effective for tasks involving text generation based on context, such as summarization, translation, and question answering.

On the other hand, Decoder-only LLMs exclusively employ the decoder module to generate target output text, focusing on predicting the next word in a sentence. These models are adept at downstream tasks with limited examples or simple instructions, as they do not require additional prediction heads or fine-tuning. Several state-of-the-art LLMs, including Llama-2 [46], Alpaca[1] , and Vicuna[2] , follow the decoder-only architecture and have achieved comparable performance with models like ChatGPT [33] and GPT-4.

3.11 Cosine Similarity

Cosine Similarity is a fundamental concept in mathematics and data analysis, particularly in the realm of information retrieval and NLP. It provides a measure of similarity between two non-zero vectors in an inner product space, typically high-dimensional spaces such as those representing documents or vectors in machine learning models.

At its core, Cosine Similarity quantifies the cosine of the angle between two vectors, thus assessing the similarity of their orientations regardless of their magnitudes. This property makes it particularly useful in scenarios where the magnitude of the vectors is less relevant compared to their direction.

For instance, in recommendation systems, Cosine Similarity is utilized to gauge the likeness between items or users based on their characteristics or behaviors. By representing

[1] https://crfm.stanford.edu/2023/03/13/alpaca.html

[2] https://lmsys.org/blog/2023-03-30-vicuna/

items or users as vectors in a high-dimensional space and computing the cosine of the angle between them, the algorithm determines their similarity. This similarity measure enables recommendation systems to suggest items that are most similar to those a user has interacted with or liked in the past, thus enhancing the overall user experience.

Given two n-dimensional vectors of attributes, **A** and **B**, the cosine similarity, $\cos(\theta)$, is represented using a dot product and magnitude as

$$\text{cosine similarity} := \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \cdot \sqrt{\sum_{i=1}^{n} B_i^2}}$$

where A_i and B_i are the i-th components of vectors **A** and **B**, respectively.

The resulting similarity ranges from -1 meaning exactly opposite, to 1 meaning exactly the same, with 0 indicating orthogonality or decorrelation, while in-between values indicate intermediate similarity or dissimilarity.

Chapter 4

Proposed Solution

As mentioned in Chapter 2, the integration of CLIP with PhoBERT offers a promising avenue for enhancing the understanding of multimodal data in the Vietnamese language context. This chapter presents the proposed solutions for a retrieval-augmented generation system aimed at retrieving the most relevant products that align with user input.

4.1 Dataset

4.1.1 Introduction

There are limited options available in Vietnamese. Crossmodal-3600 [44] introduced two multilingual datasets, COCO-35L and CC3M-35L, by translating Conceptual Captions 3M [40] and COCO Captions [6] into 34 other languages using Google's machine translation API. COCO-35L consists of $113,287$ images for training and $5,000$ images for validation, with each image having 5 reference captions. CC3M comprises $3,318,270$ image-text pairs for training and $15,840$ image-text pairs for validation.

UIT-OpenViIC [4] introduced a dataset containing complex scenes captured in Vietnam and manually annotated by native speakers under strict rules and supervision. It includes $13,100$ images, each with an average of 4.7 captions.

Recently, KTVIC [34] introduced a comprehensive Vietnamese Image Captioning dataset focused on daily life activities. This dataset comprises $4,327$ images and $21,635$ Vietnamese captions, serving as a valuable resource for advancing image captioning in the

Vietnamese language.

4.1.2 Dataset Segmentation

As mentioned, the PhoBERT tokenizer requires word and sentence segmentation on input, for which we will use the RDRSegmenter. Therefore, before training, we will segment the captions in the dataset using this segmentation tool.
The training set of UIT-OpenViIC consists of $9,088$ images with $41,238$ captions, the validation set has $2,011$ images with $10,002$ captions, and the test set comprises $2,001$ images with $10,001$ captions.

4.2 PhoCLIP

We introduce PhoCLIP, a novel approach that combines the strengths of CLIP and PhoBERT to create a powerful multimodal retrieval-augmented generation system tailored for the Vietnamese language context. PhoCLIP builds upon the foundation of CLIP's multimodal representation learning, which enables it to understand and associate images with text in a shared embedding space. By integrating PhoBERT, a state-of-the-art text encoder designed specifically for the Vietnamese language, PhoCLIP enhances its ability to comprehend and interpret Vietnamese text input.
We adopt the same design as OpenAI CLIP. The CLIP model comprises a text encoder and an image encoder, followed by a projection layer that maps their hidden states to the same multimodal space, typically with a dimensionality of 512.

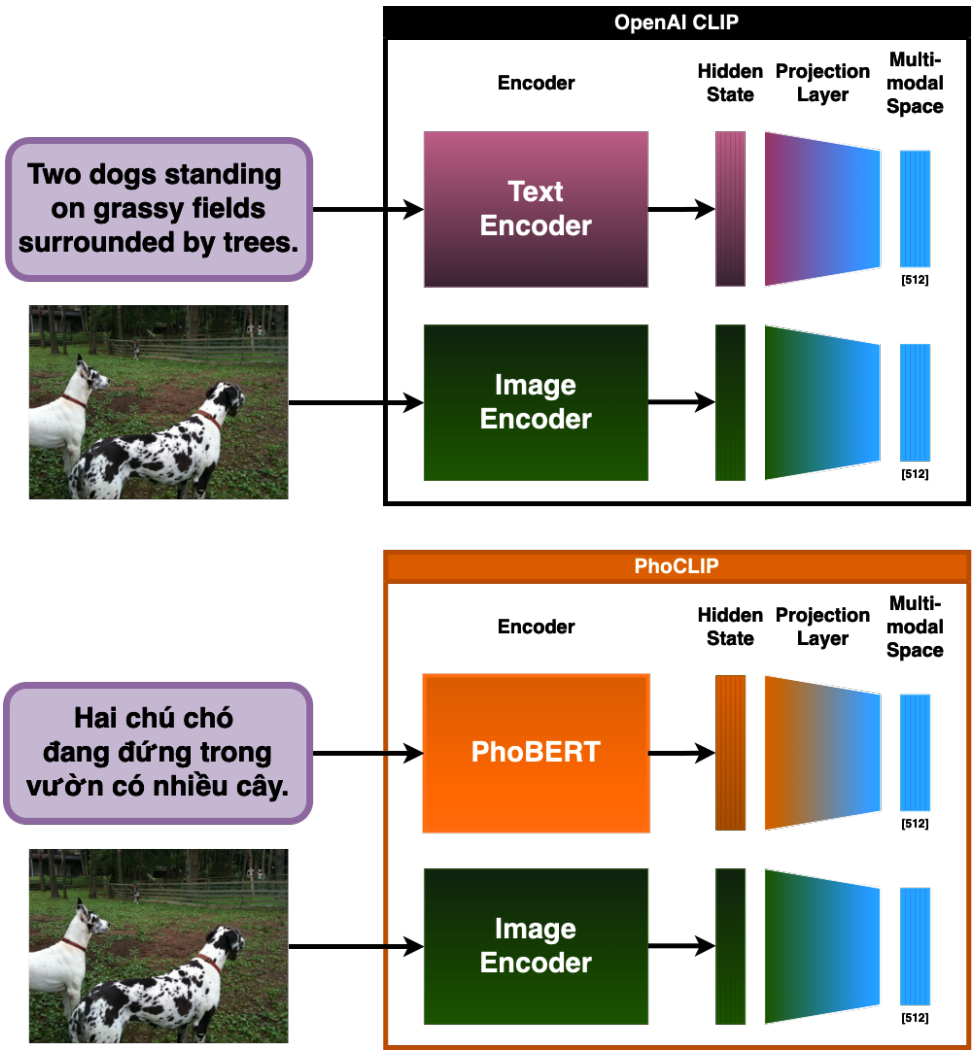


Figure 4.1: Architecture of PhoCLIP, adopt from OpenAI CLIP.

In our implementation, we replace the original text encoder with the PhoBERT encoder tokenizer, utilizing its pretrained weights and tokenizer. For the image encoder, we experiment with two popular architectures: Vision Transformer and ResNet. The projection layer is implemented as a linear layer. While CLIP uses a single linear layer for projection, we explore the effectiveness of using one and two linear layers in our experiments. By integrating PhoBERT with CLIP's architecture, PhoCLIP is capable of understanding and associating Vietnamese text with images, enabling accurate retrieval of relevant images based on textual input.

4.3 Experiments

4.3.1 Settings

We conducted four experimental models, each utilizing PhoBERT-base as the text encoder. The vision encoders used in these experiments were frozen, meaning that only the PhoBERT component and the projection layers were trained. We experimented with both ViT-Base and ResNet50 as the frozen vision encoders, following a similar approach to LiT, with variations in the number of projection layers. The models were trained for 1 epoch due to no reduction in train loss. The model size configuration is shown in Table 4.1.

Table 4.1: Experimental models size

Vision Encoder	**Linear Layers**	**Parameters**
ResNet50	1	160M
ResNet50	2	161M
ViT-B	1	222M
ViT-B	2	223M

We also utilize the COCO-35L, UIT-OpenViIC, and KTVIC datasets for both training and testing our models. The CC3M-35L dataset will be reserved for future development due to its large size. The total image-text pairs trained is about $666,518$ pairs.

4.3.2 Results

Training Results

Our training result can be referred from Table 4.2

Table 4.2: Training results

Vision Encoder	Linear Layers	Train Loss	Validation Loss
ResNet50	1	**0.52**	**2.13**
ResNet50	2	0.858	2.35
ViT-B	1	1.03	2.16
ViT-B	2	1.809	2.489

Testing Results

We verify the performance of PhoCLIP across the datasets using two metrics: top-5 accuracy and average cosine similarity (Che et al. 2024).

Top-5 accuracy evaluates the model's performance by checking, for each text-image pair in the test set, whether the top 5 most similar vectors of the embedded representation of the text contain the true image's embedded representation.

We denote:

- x_i: Embedded representation of the i-th text.
- y_i^*: True embedded representation of the image corresponding to the i-th text.

The top-5 accuracy can be formulated as follows:

$$\text{Top-5 accuracy} = \frac{1}{N}\sum_{i=1}^{N} \text{Top5}(x_i, y_i^*) \tag{4.1}$$

where:

$$\text{Top5}(x_i, y_i^*) = \begin{cases} 1 & \text{if } y_i^* \in \text{Top-5 similar images of } x_i \\ 0 & \text{otherwise} \end{cases} \tag{4.2}$$

Average cosine similarity assesses the model's performance by calculating the average cosine similarity of each text-image pair representation in the test set. Cosine similarity measures the cosine of the angle between two vectors and is used to assess the similarity between them. Higher average cosine similarity values indicate more robust semantic alignment and enhanced image-to-text transformation. If we denote:

- text_i: Embedded representation of the i-th text.

- image$_i$: Embedded representation of the i-th image.
- N: Total number of text-image pairs in the test set.

then, the average cosine similarity can be formulated as follows:

$$\text{Average cosine similarity} = \frac{1}{N}\sum_{i=1}^{N} \text{cos\_sim}(\text{text}_i, \text{image}_i)$$

where cos_sim(text$_i$, image$_i$) represents the cosine similarity between the embedded representations of text text$_i$ and image image$_i$.

The results evaluated for our metrics are shown in Table 4.3 and Table 4.4:

Table 4.3: Top 5 accuracy

Vision Encoder	**Linear Layers**	**KTVIC**	**OpenViIC**	**COCO-35L**
ResNet50	1	**0.368**	**0.269**	**0.383**
ResNet50	2	0.290	0.189	0.302
ViT-B	1	0.296	0.198	0.285
ViT-B	2	0.223	0.127	0.211

Table 4.4: Average cosine similarity

Vision Encoder	**Linear Layers**	**KTVIC**	**OpenViIC**	**COCO-35L**
ResNet50	1	**0.302**	**0.283**	**0.215**
ResNet50	2	0.155	0.141	0.057
ViT-B	1	0.262	0.254	0.202
ViT-B	2	0.148	0.112	0.052

Following the testing results, we observe that the PhoCLIP model with ResNet50 and 1 linear layer for the projection head performs the best across the test sets. Although the top 5 accuracy is not as high as the results from the original CLIP model on English datasets, PhoCLIP shows promising potential with a top 5 accuracy of up to 0.38 for the test dataset. While the top 5 accuracy may not be significant, in Appendix A, we conduct several searches using text and image queries to find relevant images and find that the PhoCLIP model with ResNet50 and 1 linear layer for the projection head can successfully retrieve suitable images given the queries.

From the experiments, we observe that increasing the number of linear layers in the projection head does not lead to better results; the original single linear layer proposed in the original OpenAI CLIP paper performs the best.

Therefore, we select the architecture of CLIP using PhoBERT as the text encoder, ResNet50 as the visual encoder, and a single linear layer for the projection head for future development. We plan to unfreeze the image encoder and train both encoders with contrastive tuning. This approach allows the model to inherit from the pretrained PhoCLIP through LiT method and effectively transfer to domain-specific data through contrastive tuning.

4.4 RAG using PhoCLIP

4.4.1 How Vector Database Is Used

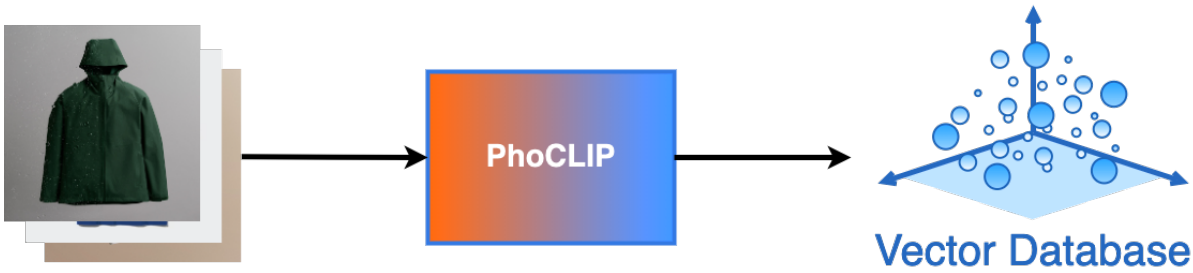


Figure 4.2: Vector Database

To efficiently store and manage our product data, we employ a Vector Database system as depicted in Figure 4.2. This database serves as the backbone of our system, facilitating the storage and retrieval of information crucial for our inferencing pipeline.

Our approach involves embedding images into vector representations using the PhoCLIP encoder. The vector representations, along with additional information such as product name, price, description, size, color, discount, etc., are then stored in our Vector Database. By including rich data alongside image vectors, we enable comprehensive and efficient retrieval of relevant information during the inferencing process.

4.4.2 Inferencing Pipeline

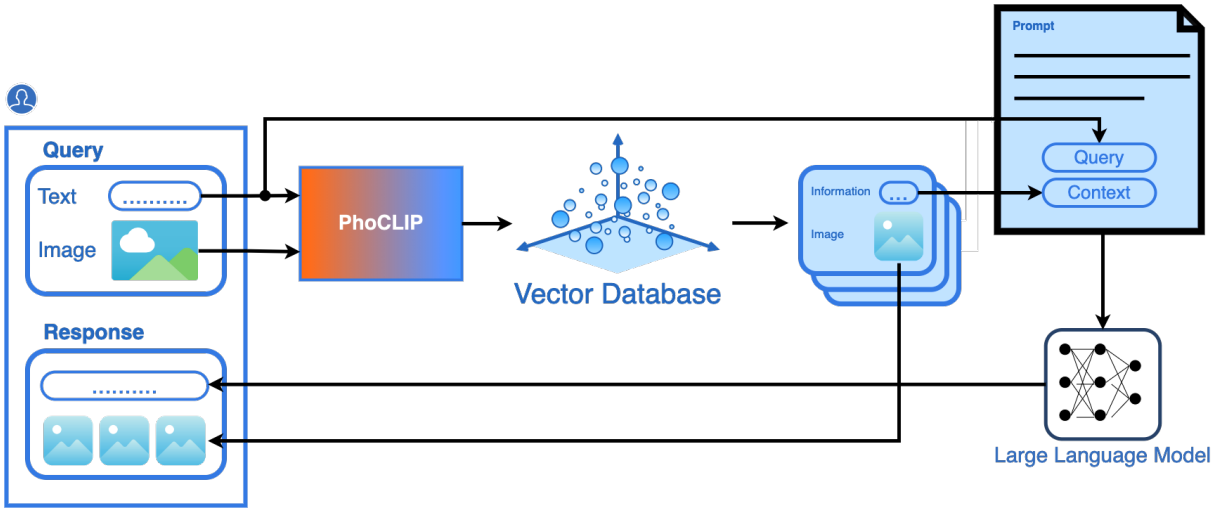


Figure 4.3: RAG using PhoCLIP

Our inferencing pipeline, illustrated in Figure 4.3, leverages RAG model empowered by PhoCLIP. The pipeline begins with a query consisting of an image and text from the user, representing what the user want the product looks like.

The PhoCLIP encoder is then utilized to convert this multimodal input into vector representations, leveraging its fine-tuned understanding of Vietnamese language. These vectors are employed to retrieve relevant contexts from the RAG Database, which houses both vector representations and the corresponding images, along with detailed information about the products, such as name, price, description, size, color, discount, and more. The retrieved context, combined with the original query, is then fed into a Vietnamese LLM, which generates detailed responses tailored to the user's needs along with the corresponding images, providing comprehensive and personalized results to enhance the user experience.

By integrating PhoCLIP's multimodal understanding with the rich information stored in the RAG Database, our inferencing pipeline delivers robust and contextually relevant responses, catering to the diverse needs of our users in the Vietnamese market.

Furthermore, we consider BARTPho as the initial LLM for RAG system of our project. In the future, we may explore experimenting with ViT5 or even MixSUra. Our approach will involve trying, testing, and evaluating the performance of these models based on pre-

determined metrics, particularly in Phase 2 of the thesis, which focuses on future work. By starting with BARTPho, we can establish a solid foundation for our experiments and pipeline development, leveraging its fine-tuned capabilities for the Vietnamese language. As we progress, we will systematically assess the effectiveness and suitability of alternative models like ViT5 and MixSUra, aiming to identify the most suitable model for our specific objectives and requirements.

Chapter 5

Conclusion

The implementation of a multimodal chatbot presents a promising solution to the challenges faced by both consumers and fashion stores in the modern retail landscape. By harnessing the power of image and text integration, this technology offers a streamlined and efficient method for customers to explore the diverse array of products available while minimizing the time and effort required in their search. Moreover, through the application of intelligent algorithms and machine learning, the chatbot can deliver personalized favorite tailored to individual preferences, thereby enhancing the overall shopping experience. Our development of PhoCLIP model specifically tailored for the Vietnamese language, marks a significant milestone in the advancement of multimodal technologies for the Vietnamese market. As the first CLIP model designed for the Vietnamese language, PhoCLIP opens doors to new possibilities in multimodal systems tailored to the unique linguistic and cultural nuances of Vietnamese consumers.

With PhoCLIP's creation comes the opportunity to establish new benchmarks and standards for multimodal systems in the Vietnamese language. By leveraging PhoCLIP as a foundational component, developers can build upon its capabilities to create more sophisticated and accurate multimodal chatbots, that seamlessly integrate image and text understanding.

By introducing a new pipeline leveraging hybrid Multimodal-RAG, fashion retailers can automate and optimize the process of product discovery and customer assistance, resulting in a twofold benefit. Firstly, these chatbots significantly enhance customer satisfaction by providing accurate, context-aware recommendations and personalized sup-

port. Secondly, they contribute to the overall operational efficiency of fashion retailers, streamlining processes and reducing the burden on human customer service representatives. For our workload and future works on this project, please refer to Appendix B.

References

[1] Giambattista Amati. "BM25". In: *Encyclopedia of Database Systems*. Ed. by LING LIU and M. TAMER ÖZSU. Boston, MA: Springer US, 2009, pp. 257–260. ISBN: 978-0-387-39940-9. DOI: 10.1007/978-0-387-39940-9_921. URL: https://doi.org/10.1007/978-0-387-39940-9_921.

[2] Saeed Anwar, Salman Khan, and Nick Barnes. *A Deep Journey into Super-resolution: A survey*. 2020. arXiv: 1904.07523 [cs.CV].

[3] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. "Latent dirichlet allocation". In: *J. Mach. Learn. Res.* 3.null (Mar. 2003), pp. 993–1022. ISSN: 1532-4435.

[4] Doanh C. Bui, Nghia Hieu Nguyen, and Khang Nguyen. *UIT-OpenViIC: A Novel Benchmark for Evaluating Image Captioning in Vietnamese*. 2023. arXiv: 2305.04166 [cs.CV].

[5] Chang Che et al. *Enhancing Multimodal Understanding with CLIP-Based Image-to-Text Transformation*. 2024. arXiv: 2401.06167 [cs.CV].

[6] Xinlei Chen et al. *Microsoft COCO Captions: Data Collection and Evaluation Server*. 2015. arXiv: 1504.00325 [cs.CV].

[7] Yen-Chun Chen et al. "Uniter: Universal image-text representation learning". In: *ECCV*. 2020.

[8] Kevin Clark et al. "Pre-Training Transformers as Energy-Based Cloze Models". In: *EMNLP*. 2020. URL: https://www.aclweb.org/anthology/2020.emnlp-main.20.pdf.

[9] Jacob Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *arXiv preprint arXiv:1810.04805* (2018).

[10] Vu-Thuan Doan et al. *Efficient Finetuning Large Language Models For Vietnamese Chatbot*. 2023. arXiv: 2309.04646 [cs.CL].

[11] Alexey Dosovitskiy et al. "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale". In: *ICLR* (2021).

[12] Difei Gao et al. "Multi-modal graph neural network for joint reasoning on vision and scene text". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 12746–12756.

[13] Ian Goodfellow et al. "Generative adversarial nets". In: *Advances in neural information processing systems* 27 (2014).

[14] GM Harshvardhan et al. "A comprehensive survey and analysis of generative models in machine learning". In: *Comput. Sci. Rev.* 38 (2020), p. 100285. URL: https://api.semanticscholar.org/CorpusID:224930744.

[15] Edward J Hu et al. "LoRA: Low-Rank Adaptation of Large Language Models". In: *International Conference on Learning Representations*. 2022. URL: https://openreview.net/forum?id=nZeVKeeFYf9.

[16] Gao Huang et al. *Densely Connected Convolutional Networks*. 2018. arXiv: 1608.06993 [cs.CV].

[17] Samuel Humeau et al. *Poly-encoders: Transformer Architectures and Pre-training Strategies for Fast and Accurate Multi-sentence Scoring*. 2020. arXiv: 1905.01969 [cs.CL].

[18] Zhi Jing et al. *When Large Language Models Meet Vector Databases: A Survey*. 2024. arXiv: 2402.01763 [cs.DB].

[19] Omar Khattab and Matei Zaharia. "Colbert: Efficient and effective passage search via contextualized late interaction over bert". In: *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval*. 2020, pp. 39–48.

[20] Douwe Kiela et al. "Supervised Multimodal Bitransformers for Classifying Images and Text". In: *arXiv preprint arXiv:1909.02950* (2019).

[21] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG].

[22] Zhenzhong Lan et al. *ALBERT: A Lite BERT for Self-supervised Learning of Language Representations*. 2020. arXiv: 1909.11942 [cs.CL].

[23] Patrick Lewis et al. *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks*. 2021. arXiv: 2005.11401 [cs.CL].

[24] Haotian Liu et al. *Improved Baselines with Visual Instruction Tuning*. 2023.

[25] Haotian Liu et al. *LLaVA-NeXT: Improved reasoning, OCR, and world knowledge*. Jan. 2024. URL: https://llava-vl.github.io/blog/2024-01-30-llava-next/.

[26] Haotian Liu et al. *Visual Instruction Tuning*. 2023.

[27] Yinhan Liu et al. *RoBERTa: A Robustly Optimized BERT Pretraining Approach*. 2019. arXiv: 1907.11692 [cs.CL].

[28] Shibo Lu et al. "DA-DCGAN: An Effective Methodology for DC Series Arc Fault Diagnosis in Photovoltaic Systems". In: *IEEE Access* 7 (2019), pp. 45831–45840. DOI: 10.1109/ACCESS.2019.2909267.

[29] Dat Quoc Nguyen et al. "A Fast and Accurate Vietnamese Word Segmenter". In: *Proceedings of the 11th International Conference on Language Resources and Evaluation (LREC 2018)*. 2018, pp. 2582–2587.

[30] Thai Binh Nguyen et al. "Improving Vietnamese Named Entity Recognition from Speech Using Word Capitalization and Punctuation Recovery Models". In: *Proc. Interspeech 2020*. 2020, pp. 4263–4267. DOI: 10.21437/Interspeech.2020-1896.

[31] Rodrigo Nogueira and Kyunghyun Cho. *Passage Re-ranking with BERT*. 2020. arXiv: 1901.04085 [cs.IR].

[32] Myle Ott et al. "fairseq: A Fast, Extensible Toolkit for Sequence Modeling". In: *Proceedings of NAACL-HLT 2019: Demonstrations*. 2019.

[33] Long Ouyang et al. *Training language models to follow instructions with human feedback*. 2022. arXiv: 2203.02155 [cs.CL].

[34] Anh-Cuong Pham et al. *KTVIC: A Vietnamese Image Captioning Dataset on the Life Domain*. 2024. arXiv: 2401.08100 [cs.CV].

[35] Long Phan et al. "ViT5: Pretrained Text-to-Text Transformer for Vietnamese Language Generation". In: *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies: Student Research Workshop*. Association for Computational Linguistics, 2022, pp. 136–142. URL: https://aclanthology.org/2022.naacl-srw.18.

[36] Alec Radford et al. *Learning Transferable Visual Models From Natural Language Supervision*. 2021. arXiv: 2103.00020 [cs.CV].

[37] Colin Raffel et al. "Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer". In: *Journal of Machine Learning Research* 21.140 (2020), pp. 1–67. URL: http://jmlr.org/papers/v21/20-074.html.

[38] "TF–IDF". In: *Encyclopedia of Machine Learning*. Ed. by Claude Sammut and Geoffrey I. Webb. Boston, MA: Springer US, 2010, pp. 986–987. ISBN: 978-0-387-30164-8. DOI: 10.1007/978-0-387-30164-8_832. URL: https://doi.org/10.1007/978-0-387-30164-8_832.

[39] Rico Sennrich, Barry Haddow, and Alexandra Birch. *Neural Machine Translation of Rare Words with Subword Units*. 2016. arXiv: 1508.07909 [cs.CL].

[40] Piyush Sharma et al. "Conceptual Captions: A Cleaned, Hypernymed, Image Alt-text Dataset For Automatic Image Captioning". In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Ed. by Iryna Gurevych and Yusuke Miyao. Melbourne, Australia: Association for Computational Linguistics, July 2018, pp. 2556–2565. DOI: 10.18653/v1/P18-1238. URL: https://aclanthology.org/P18-1238.

[41] Xinying Song et al. *Fast WordPiece Tokenization*. 2021. arXiv: 2012.15524 [cs.CL].

[42] Hao Tan and Mohit Bansal. "LXMERT: Learning Cross-Modality Encoder Representations from Transformers". In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. 2019.

[43] Rohan Taori et al. *Stanford Alpaca: An Instruction-following LLaMA model*. https://github.com/tatsu-lab/stanford_alpaca. 2023.

[44] Ashish V. Thapliyal et al. *Crossmodal-3600: A Massively Multilingual Multimodal Evaluation Dataset*. 2022. arXiv: 2205.12522 [cs.CV].

[45] Ilya Tolstikhin et al. "MLP-Mixer: An all-MLP Architecture for Vision". In: *arXiv preprint arXiv:2105.01601* (2021).

[46] Hugo Touvron et al. *Llama 2: Open Foundation and Fine-Tuned Chat Models*. 2023. arXiv: 2307.09288 [cs.CL].

[47] Nguyen Luong Tran, Duong Minh Le, and Dat Quoc Nguyen. "BARTpho: Pre-trained Sequence-to-Sequence Models for Vietnamese". In: *Proceedings of the 23rd Annual Conference of the International Speech Communication Association*. 2022.

[48] Sang T. Truong et al. *Crossing Linguistic Horizons: Finetuning and Comprehensive Evaluation of Vietnamese Large Language Models*. 2024. arXiv: 2403.02715 [cs.CL].

[49] Ashish Vaswani et al. *Attention Is All You Need*. 2023. arXiv: 1706.03762 [cs1.CL].

[50] Thanh Vu et al. "VnCoreNLP: A Vietnamese Natural Language Processing Toolkit". In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Demonstrations*. Ed. by Yang Liu, Tim Paek, and Manasi Patwardhan. New Orleans, Louisiana: Association for Computational Linguistics, June 2018, pp. 56–60. DOI: 10.18653/v1/N18-5012. URL: https://aclanthology.org/N18-5012.

[51] Saining Xie et al. *Aggregated Residual Transformations for Deep Neural Networks*. 2017. arXiv: 1611.05431 [cs.CV].

[52] An Yang et al. "Chinese CLIP: Contrastive Vision-Language Pretraining in Chinese". In: *arXiv preprint arXiv:2211.01335* (2022).

[53] Zhilin Yang et al. *XLNet: Generalized Autoregressive Pretraining for Language Understanding*. 2020. arXiv: 1906.08237 [cs.CL].

[54] Sergey Zagoruyko and Nikos Komodakis. *Wide Residual Networks*. 2017. arXiv: 1605.07146 [cs.CV].

[55] Xiaohua Zhai et al. “LiT: Zero-Shot Transfer with Locked-image Text Tuning”. In: *CVPR* (2022).

[56] Xiaohua Zhai et al. *Sigmoid Loss for Language Image Pre-Training*. 2023. arXiv: 2303.15343 [cs.CV].

Appendix A

Inference on PhoCLIP

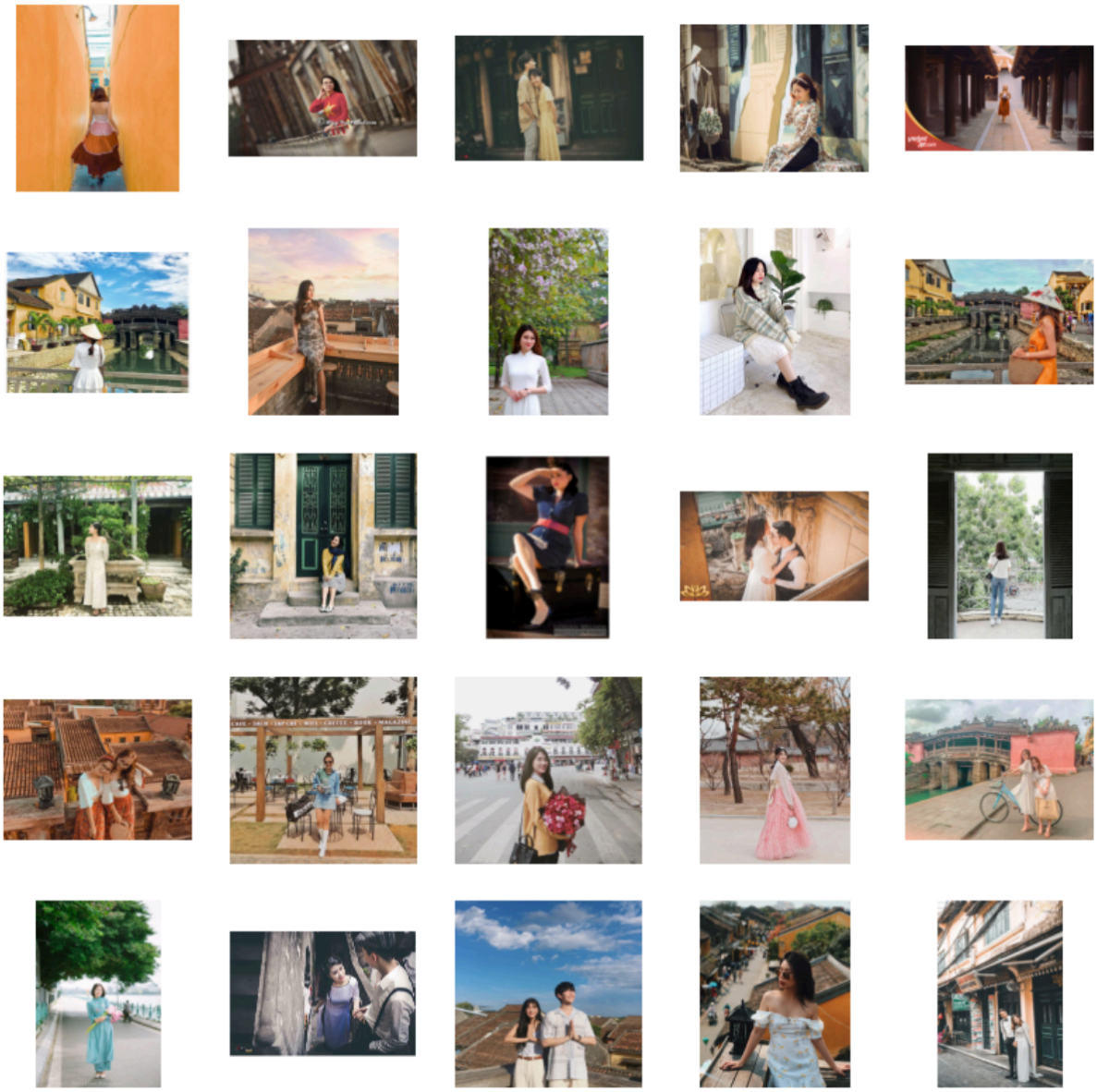
Below, we showcase our inference results on query images from the test set using text queries, image queries, and combination of 2 modalities. Each query represents a specific visual concept, and our system retrieves the top matching images based on the provided text description.

A.1 Text-only queries

Here are some inference results when we query our model on by text, to test text embedding performance:

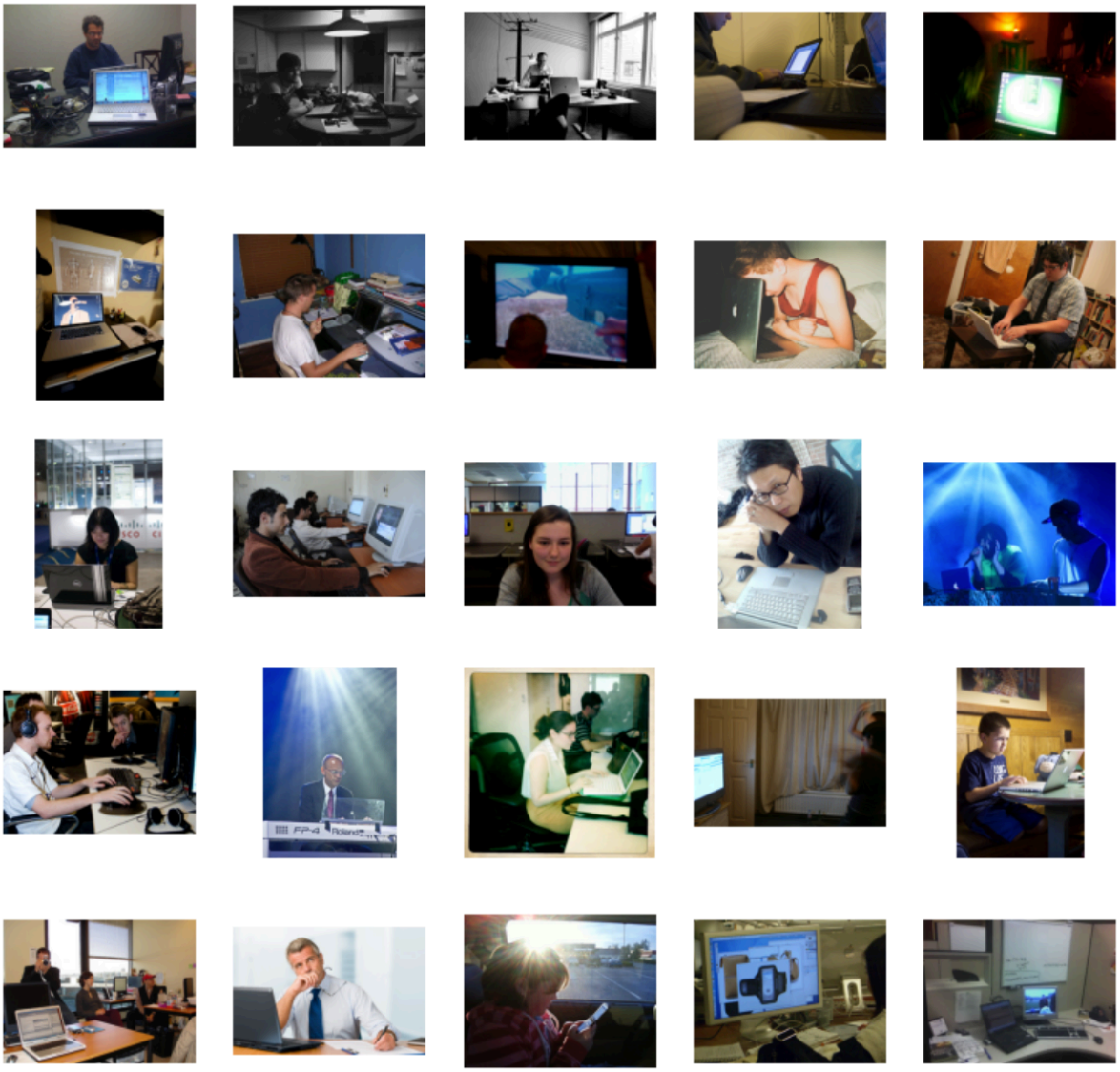
- **Text query:** *”cô gái mặc áo dài”* (a girl in Ao Dai)

  Top matches:



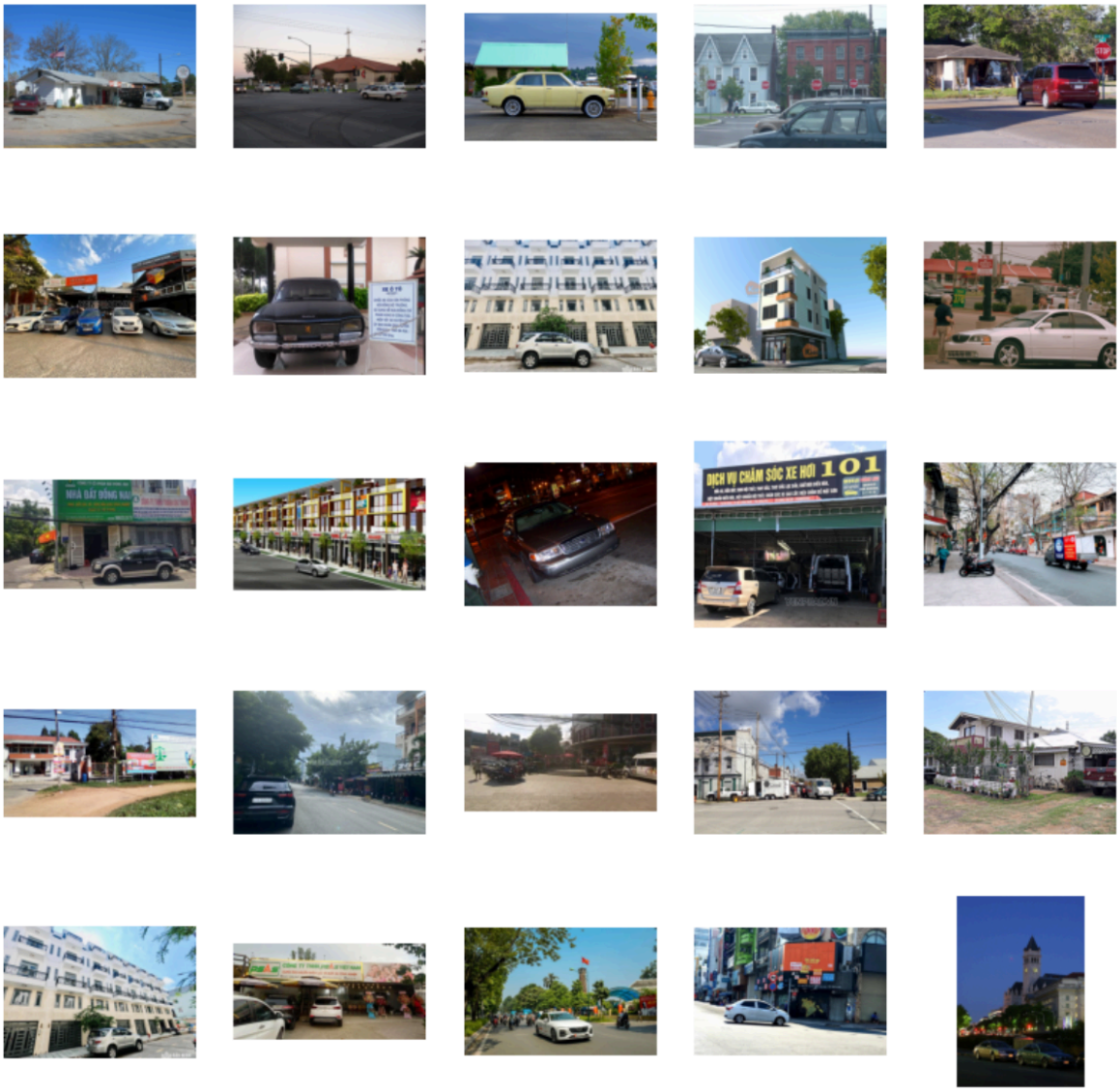
- **Text query:** *"một người đang dùng máy tính"* (a person using a computer)

  Top matches:



- **Text query:** *xe hơi đậu trước ngôi nhà* (a car parks in front of a house)

  Top matches:



- **Text query:** *em bé chơi đùa cùng đồ chơi* (a baby plays with toys)

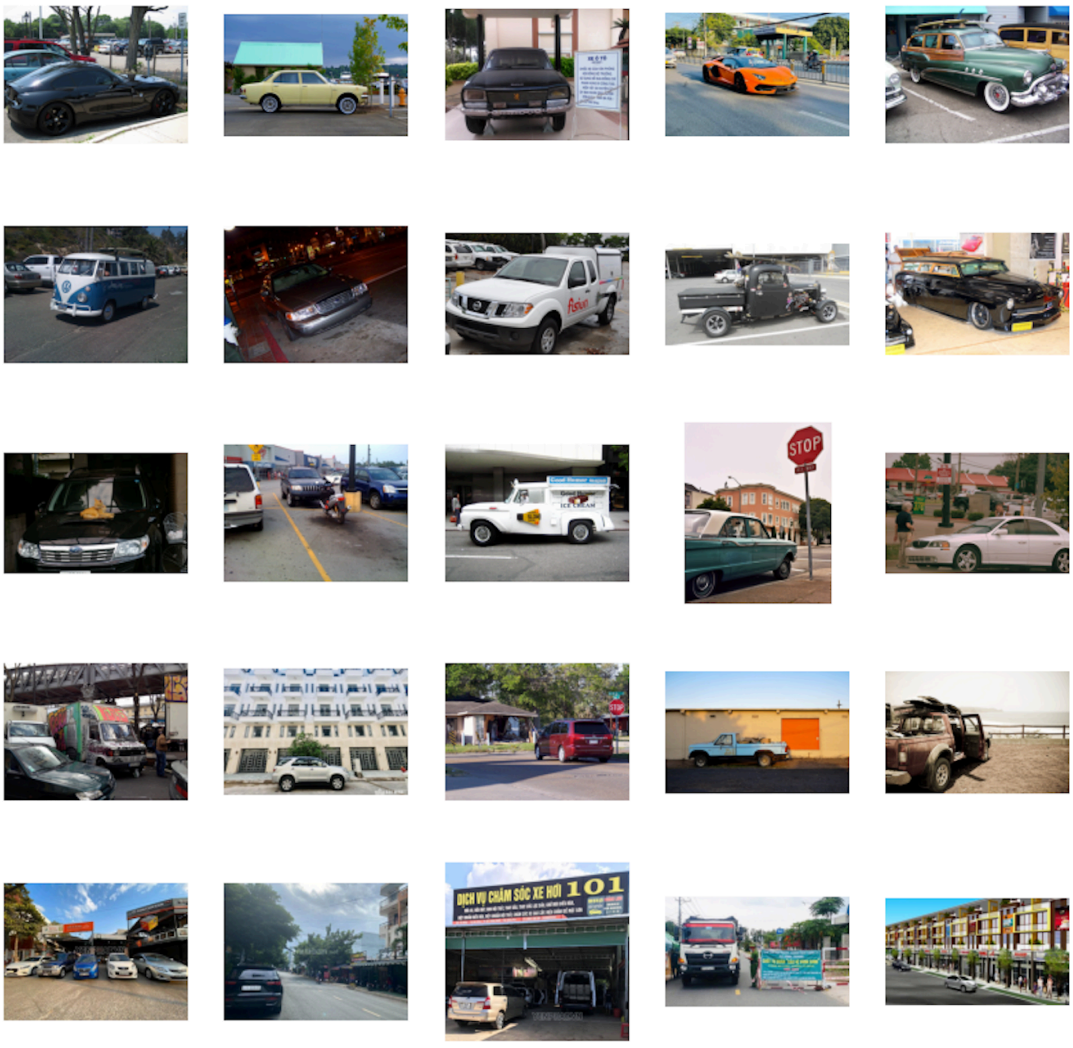
  Top matches:

A.2 Image-only queries

Here are some inference results when we query our model on by images, to test image embedding performance:

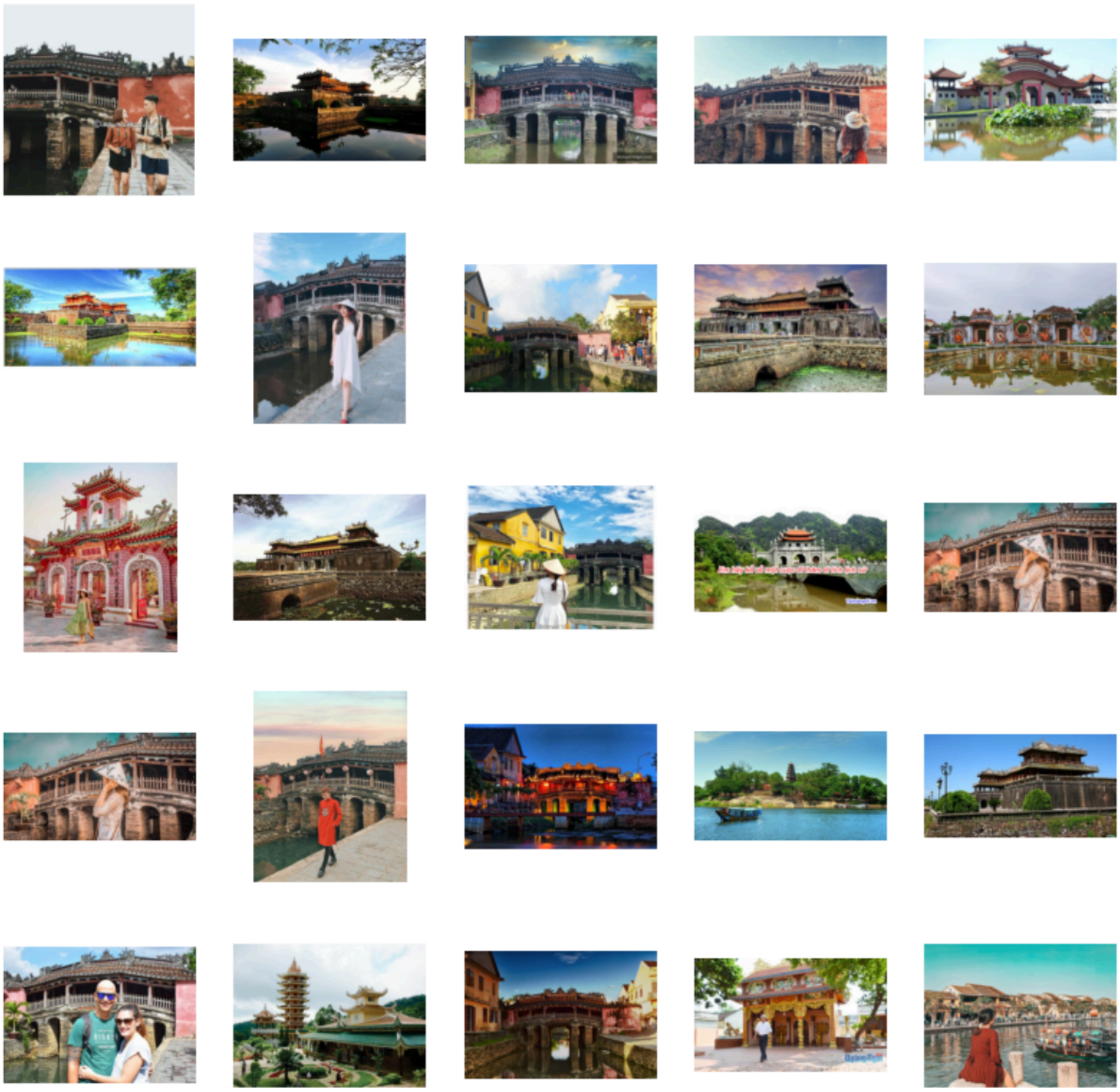
- **Image query:**

Top matches:



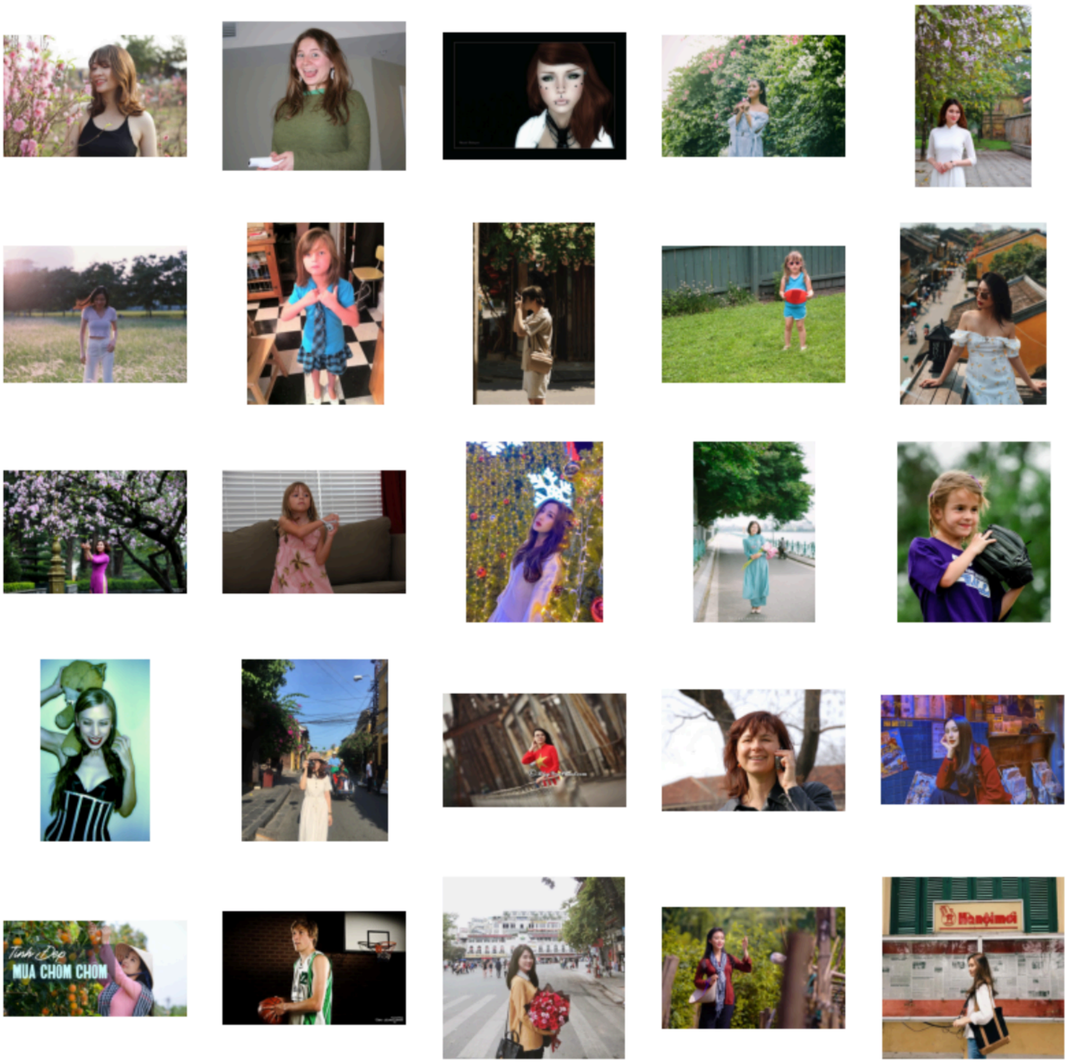
- **Image query:**

Top matches:

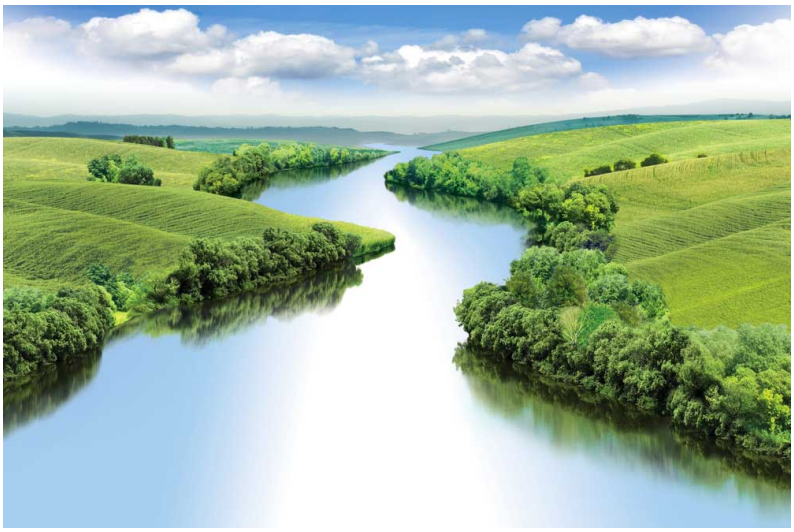


- **Image query:**

Top matches:



- **Image query:**



Top matches:























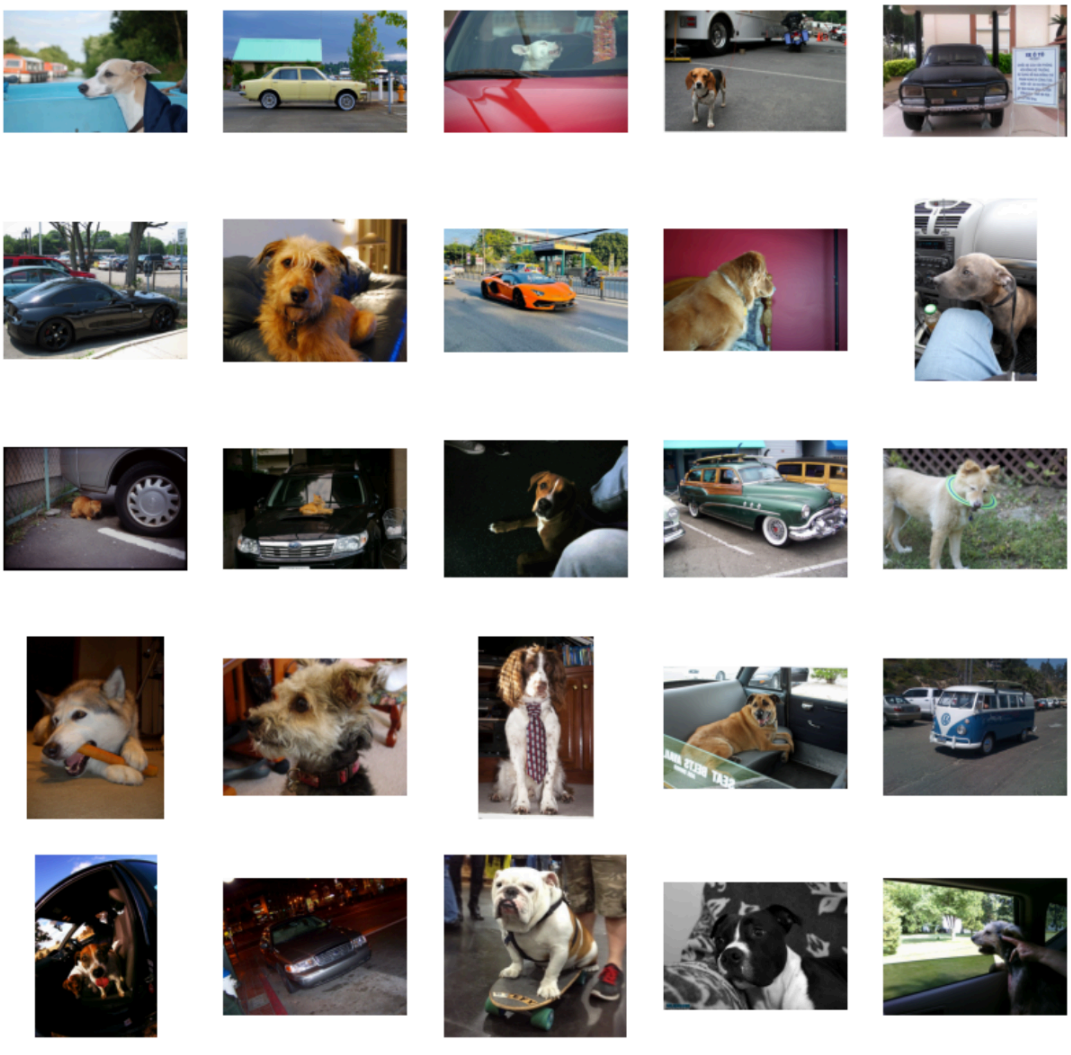

A.3 Image-text combined queries

Some results when we combined image and text prompt for queries:

- **Image query:**

Text query: *"chú chó vui vẻ"* (a happy dog)

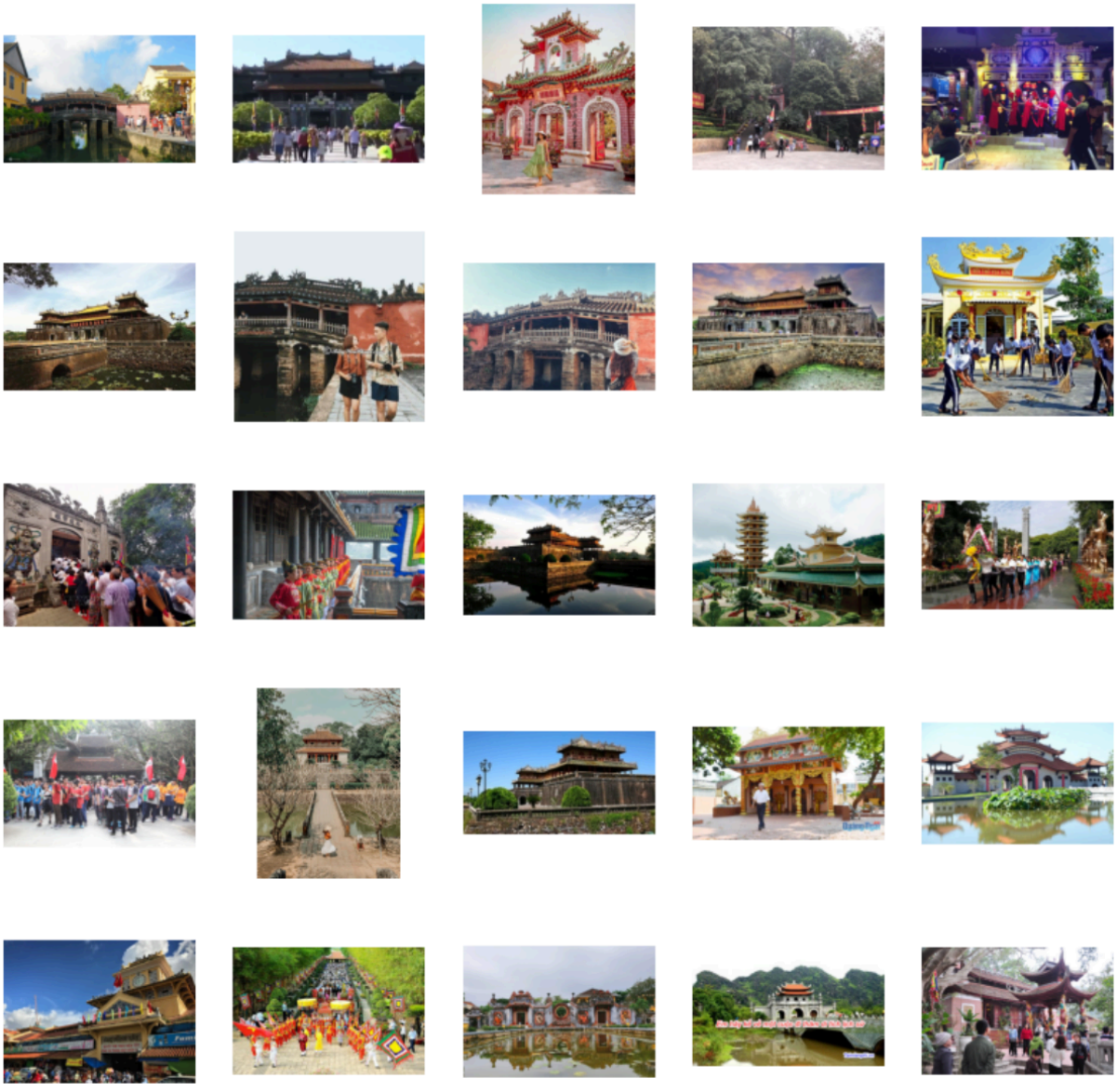
Top Matches:



- **Image query:**

Text query: *"nhiều người tham quan"* (many people visit)

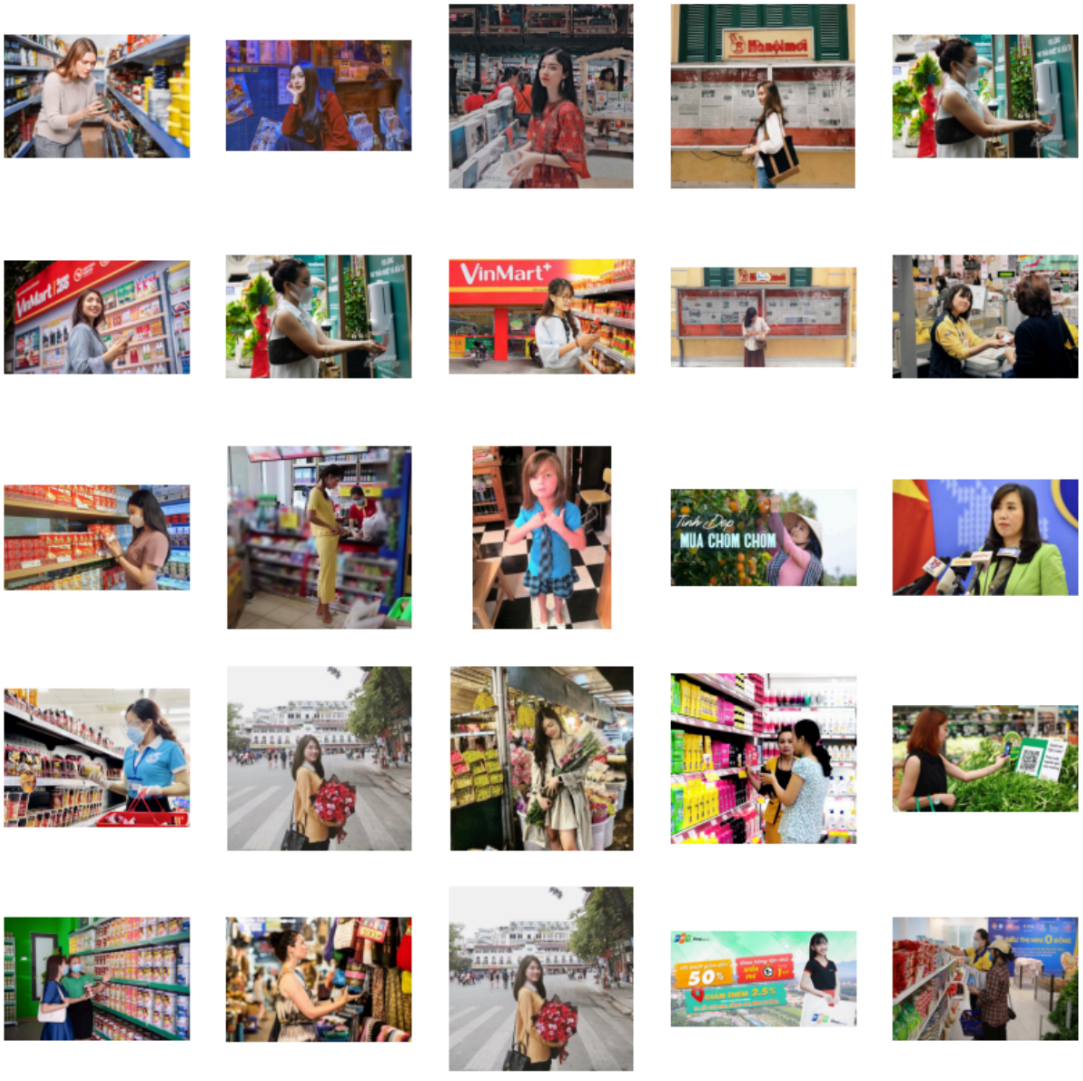
Top matches:



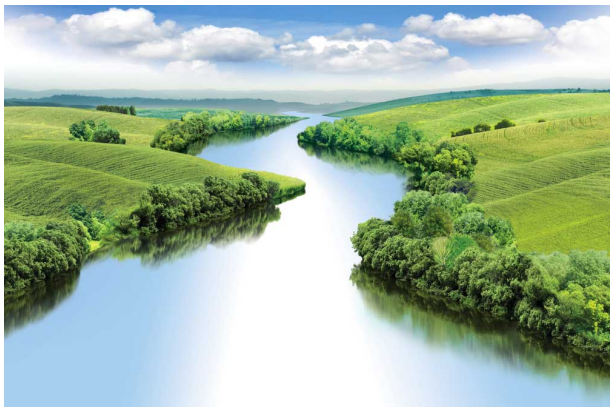
- **Image query:**

Text query: *"đi siêu thị"* (go to supermarket)

Top matches:



- **Image query:**



- **Text query:** *”thành phố”* (a city)

  Top matches:
























Appendix B

Table of Workload

B.1 Phase 1: Done

Work	**In charge**
Research on proposed pipeline	Nhat Khang, Tri Duc
Research on datasets for training	Nhat Khang, Tri Duc
Research on CLIP model and related works	Tri Duc
Research on RAG and related works	Nhat Khang
Training and evaluating PhoCLIP	Tri Duc
Theoretical knowledge preparation for the project	Nhat Khang

B.2 Phase 2: Coming up

In Phase 2 of our project, we will focus on several key tasks aimed at enhancing the capabilities and performance of our multimodal inference system. Specifically, we will explore methods for hybridizing information derived from both text embedding and image embedding within the CLIP framework. This research aims to enhance the synergy between textual and visual information in our multimodal inference system, ultimately leading to more nuanced and contextually rich responses. By delving into the potential of information encoding from both modalities, we anticipate further advancements in

the system's ability to comprehend and generate responses based on diverse multimodal inputs.

The design and implementation of a Vector Database will be crucial for efficiently storing and managing the vector representations of images and associated metadata. This database will serve as the backbone of our system, enabling quick and effective retrieval of relevant information during the inferencing process.

Next, the implementation of the Retrieval-Augmented Generation (RAG) model will be a pivotal step in our project. RAG integrates information retrieval with text generation, allowing for more contextually relevant and coherent responses. Connecting PhoCLIP, our pretrained CLIP model fine-tuned for the Vietnamese language, with RAG will further enhance the system's ability to understand and generate responses based on multimodal inputs, thereby enabling responses with both text and images.

Additionally, we will embark on the task of crawling fashion data from various fashion websites. This data will be used for further fine-tuning and testing of the RAG system and the entire pipeline. By incorporating real-world fashion data into our system, we aim to improve its accuracy and relevance in providing fashion-related recommendations and assistance.

Once the system components are in place, extensive system testing will be conducted to ensure functionality, reliability, and performance under various scenarios and conditions. This testing phase will involve rigorous evaluation of individual components as well as the integrated system as a whole. Comprehensive testing protocols will be devised to assess the system's robustness and accuracy across a diverse range of inputs and use cases.

Finally, the overall results of the system will be evaluated based on predetermined metrics and criteria. These evaluations will provide insights into the system's effectiveness, efficiency, and suitability for real-world applications, guiding further iterations and refinements to optimize performance and user satisfaction.