FEDERAL STATE AUTONOMOUS EDUCATIONAL INSTITUTION
HIGHER EDUCATION «NATIONAL RESEARCH UNIVERSITY
«HIGH SCHOOL OF ECONOMICS»

Faculty of Informatics, Mathematics and Computer Science

**The program of preparation of bachelors in the direction
01.03.02 Applied mathematics and informatics**

Schenikova Anna Yurievna

GRADUATE QUALIFICATION THESIS

Poetry and music generation from images using NLP and CV tools

Reviewer

Senior Lecturer,

Department of Applied Mathematics and Informatics, National Research University Higher School of Economics - Nizhny Novgorod

Dosov Sanjar Muzaffarovich

Scientific adviser

Guest Lecturer,

Department of Applied Mathematics and Informatics, National Research University Higher School of Economics - Nizhny Novgorod

Shimko Alexey Andreevich

Nizhny Novgorod, 2023

Contents

Introduction

In an age where artificial intelligence (AI) has shown remarkable potential in a wide variety of fields, exploring its applications in creative domains such as music and poetry has broaden the horizons. This intersection of creativity and technology raises burning questions about the ability of machines to replicate human creativity and create emotional, inspiring art. This diploma thesis aims to contribute to this emerging field by focusing on the creation of music and poetry based on a visual picture of the world using Natural Language Processing (NLP) and Computer Vision (CV) tools.

0.1 Formulation of the problem

Despite advances in artificial intelligence, creating meaningful poetry and music from visual data remains a major challenge. From the point of view of poetry, the task is to create semantically coherent, linguistically rich and emotionally resonant poems. For music, the complexity increases with the need to create not only melodic, but also rhythmically consistent and aesthetically pleasing pieces.

Existing solutions lack variety and emotional depth and are often unable to effectively encapsulate the nuances of the themes and feelings expressed in an image. This thesis project aims to explore and solve this problem by formulating a solution that uses state-of-the-art NLP and CV tools.

0.2 Relevance of the topic

The relevance of the topic is underlined by the surge of interest in art created by artificial intelligence, which has serious consequences for artists, the creative industry and society as a whole. The study of AI in creative spaces can lead to new architectures and solutions in bridging the gap between different sub-disciplines of AI. The ability to generate poetry and music from images not only holds the potential for artistic applications, but also expands our understanding of how machines can interpret and translate human emotions and abstract concepts.

0.3 Model object

The object of the research is the possibility of creating a system capable of generating poetry and music based on an image. The limitations of this system lie in the fact that there are a number of semantic, stylistic, rhythmic and melodic requirements for the output text, which can be called poetry or music. The possibility of generating music and poetry that reflects the essence and emotions inspired by the image is also being explored.

0.4 The development of the topic

Image captioning models, such as those based on CNN-RNN architectures (Vinyals et al. [2015]), have demonstrated the ability to generate accurate and descriptive captions for images. However, these models often generate generic and repetitive phrases, lacking creativity and artistic quality. Furthermore, these models are optimized for conciseness and informativeness rather than the emotional and expressive language often associated with poetry.

Visual storytelling models (Huang et al. [2016]) address some of these limitations by generating longer and more creative narratives. However, the coherence and artistic quality of generated narratives can be inconsistent. Both image captioning and visual storytelling models do not specifically address the challenges of generating poetry, such as adherence to poetic structures and conventions.

Briot et al. [2019] proposed a deep learning model for generating music from images, utilizing a CNN to extract image features and an LSTM-based music generation model. Although this approach showcases the potential for generating music from visual inputs, it presents certain limitations. For instance, the generated music may not consistently be contextually relevant to the input image, as the model does not explicitly account for musical structure or style. Furthermore, the model does not address poetry generation or music alignment with generated text.

Research on generating poetry from pictures remains scarce. Li et al. [2018] proposed a model generating Chinese poetry from images, using a CNN for image feature extraction and an RNN for poetry generation. Despite showcasing the potential for poetry generation from visual inputs, the model has limitations, including language specificity, potential lack of diversity and creativity, and no music generation or integration with generated poetry.

Vision Transformers (ViT) have become a powerful architecture for image recognition tasks, which means moving away from traditional CNNs for feature extraction. Originally proposed by Papineni et al. [2002], the ViT model applies a transformer architecture previously successful in natural language processing (NLP) directly to image pixels, capitalizing on its ability to capture long-term dependencies in data. The principles behind ViT can be useful in tasks such as creating poetry from images, as they allow models to understand the comprehensive storytelling and aesthetics of visual inputs.

The groundbreaking article “Attention is all you need” by Vaswani et al. [2017] presented the concept of the Transformer model, a new architecture based solely on attentional mechanisms, completely eliminating repetition and convolutions. This was a significant departure from traditional RNN sequence models, especially in NLP tasks. This principle has been transformational, leading to the creation of powerful encoder-decoder models such as GPT (the decoder in article Brown et al. [2020]) and BERT (the encoder in article Devlin et al. [2019]), which have demonstrated remarkable performance in a variety of tasks, including machine translation, text generation, and many others.

A recent article on generating music from text based on different pre-trained models such as GPT-2, BERT and BART (Lewis et al. [2019]) models, introduces the possibility of using the developed architecture for the more complex task of generating music from pictures.

0.5 The purpose of the diploma

The purpose of the diploma is to develop and evaluate an algorithm for generating poetry in the style of 5 famous Russian authors based on an image and generating music from an image using and modifying existing solutions in the field of NLP and CV.

0.5.1 Objectives

The objectives of the thesis work are:

1. Review and generalization of existing approaches and solutions for the problem at the intersection of creating music, image processing, text generation;
2. Defining model architectures that can potentially effectively transform visual inputs into poetic and musical outputs;
3. Identifying suitable training datasets and collecting them;
4. Determination of key metrics to assess the quality of the generation of poetry or music output;
5. Development and training of models for the selected architectures and evaluation of their generation quality in accordance with the metrics;
6. Analyze the ability of models to generate art that emotionally matches the input image.

Through these challenges, this diploma seeks to push the boundaries of what is currently possible in the realm of AI-generated art, offering new perspectives on how machines can contribute to creative industries and a greater understanding of the nature of humanity.

1 Theoretical explanation

The development of a system capable of generating poetry and music from images requires a deep study of existing technologies and approaches. It is necessary to evaluate independent algorithms for the automatic generation of poems and music, as well as to consider existing approaches for extracting information from images and its further use to solve the problem.

In this chapter, the following technologies and approaches will be considered in detail:

1. visial transformers for features extraction from images;
2. BERT and Sentence Transformers for keywords extraction;
3. training full-fledged image-to-text and image-to-music models;
4. solving the independent problem of generating music and poetry.

Thanks to existing developments, it has become possible to obtain interesting solutions for such a high-level problem as multimodal generation using small computing power.

1.1 Encoder-decoder models

Most language and visual models before transformers already consisted of two parts: an encoder and a the decoder. The direct function of the encoder is to generate a low-dimensional representation based on the input information, while the purpose of the decoder is to transform this low-dimensional representation in such a way as to obtain an output value comparable to the task.

As an example, the translation task can be considered: a sentence in Russian is fed to the input of the model, the encoder forms a low-level representation in the form of a vector based on it, and the decoder transforms the vector so that the output is a sentence in English.

In my term paper on generating poetry using NLP tools, I went into detail about what the encoder-decoder of the model was before transformers, and how the attention mechanism fixed the main problem of the encoder compressing the source sentence into a single vector and decoder receiving information from a small-sized representation vector, as shown in the **Figure 1.**

1.1.1 Transformer architecture

The approach to encoder-decoder models changed with the advent of **Transformers** (Vaswani et al. [2017]), as they were able to fix the problem of low-level representa-

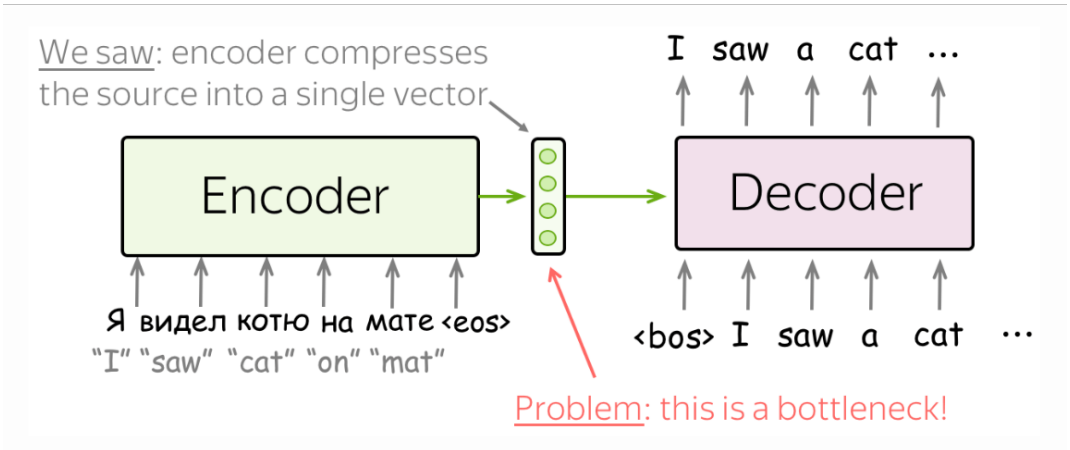


Figure 1: Problem of Encoder-Decoder architecture before Transformers.

tion from encoder to decoder and integrate different types of attention, which gave a new impetus to the development of language and visual models. For later conclusions, it is necessary to consider the architecture of the transformer. A general view of the model can be seen on **Figure 2**.

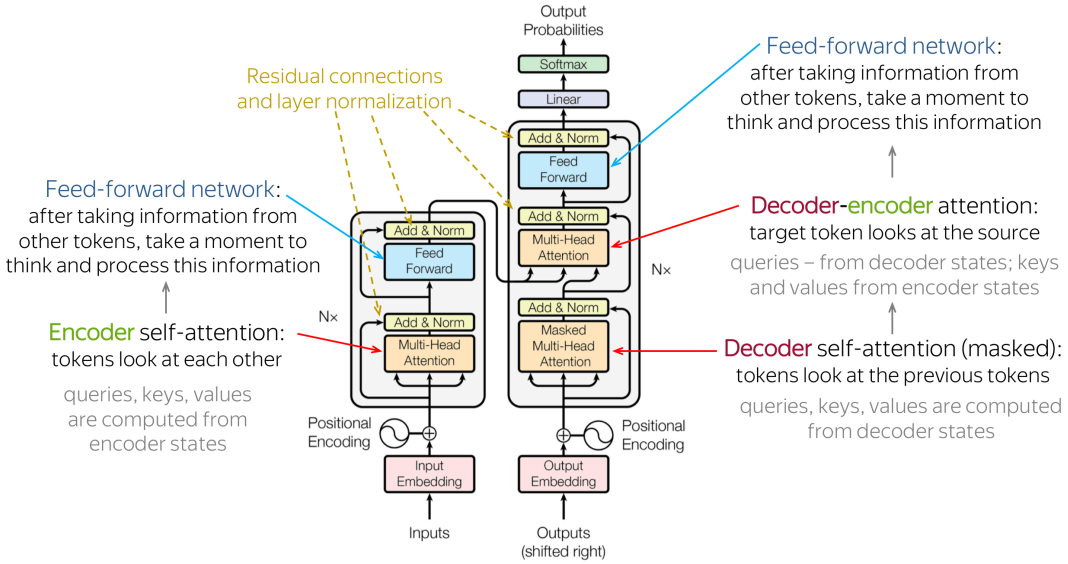


Figure 2: Transformer model in details with comments.

Form embeddings.

At the moment of transferring the original sentence in Russian to the encoder, as well as receiving the English sentence to the decoder in training process, there is an important part: tokenization and the formation of embeddings. Our human language must be translated into a language understandable to the model, for example, to form vector values that would reflect the semantics of words.

The steps are:

1. splitting the input sentence into tokens,
2. for each token obtaining dense vector representation (embedding) of length D.

So, an $N \times D$ embedding matrix is formed at the input stage, where N is the number of tokens in the input sentence. A dense vector means that most of the values in it are non-zero.

Embeddings are optimized during the training of the encoder-decoder transformer in order to better form an understanding of the information being processed.

Positional encoding.

The transformer model is designed to receive information about the input text not sequentially, but simultaneously for the entire sentence. Unlike recurrent and convolutional networks, transformers look at all tokens at once, which solves the problem of a long understanding of what this sentence is about, but information about the position of each word is lost, and this is critical for the language. Therefore, positional embeddings were introduced, which are calculated by the formula:

$$\mathrm{PE}_{pos,2i} = \sin\left(pos/10000^{2i/d_{model}}\right),$$
$$\mathrm{PE}_{pos,2i+1} = \cos\left(pos/10000^{2i/d_{model}}\right),$$

where pos - position of an object in the input sequence, d_{model} - dimension of the output embedding space and i is an integer that ranges from 0 to $d_{model}/2$ which uses to calculate a different value for each dimension of the positional encoding vector of length D with a single value of maps to both sine and cosine functions.

Multi-Head Attention.

As shown in the **Figure 3(right)**, a Multi-head attention consists of h heads into which the Scaled dot-product attention mechanism is computed using a single algorithm.

The steps are:

1. The matrix of input embeddings $\mathbf{X}$ of dimension $N \times D$ is divided into three parts: query (Q), key (K) and value (V), by multiplying the input embeddings by learning matrices W_Q, W_K, W_V, and for each head these matrices are different:

$$Q = \mathbf{X}W_Q^i, W_Q^i \in \mathbb{R}^{D \times d_k}$$
$$K = \mathbf{X}W_K^i, W_K^i \in \mathbb{R}^{D \times d_k}$$
$$V = \mathbf{X}W_V^i, W_V^i \in \mathbb{R}^{D \times d_k}$$
$$i = 1, \ldots, h,$$

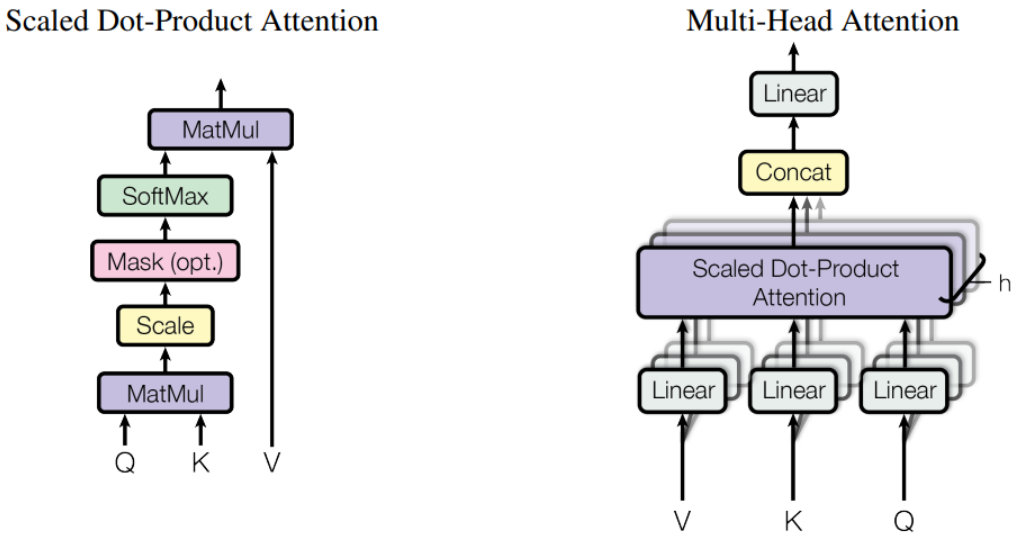


Figure 3: How self-attention and multi-head attention work. Source: Vaswani et al. [2017]

where $N \times d_k$ - is the dimensionality of the queries, keys, and values.

Different transformation matrices for query, key and value for each head contribute to a multilateral view of the mechanism of attention to the information being processed.

2. Attention is calculated as (**Figure 3 left**) and softmax function can be found in Appendix 4.4:

$$\text{Attention} = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{1}$$

$$\text{Attention} \in \mathbb{R}^{N \times d_k}$$

These parts (Q, K, V) have different representation roles:

- Q - asking for information: each token looking at others to find relevant information about himself, matrix from which attention is looking;
- K - saying that it has some information: it is necessary to indicate which tokens have relevant tokens from query's request, matrix at which query looks to compute weights;
- V - giving the information: weighted summary for each token to obtain the attention weights, which represent the degree of 'attention' given to each key-value pair.

3. Within multi-head attention, input embeddings are used to count the attention of each head, and the results obtained are concatenated and multiplied by a common transformation matrix W_o to transform the received information into the dimension of input embeddings ($N \times D$):

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_n)W_o, \tag{2}$$

$$W_o \in \mathbb{R}^{hd_k \times D}$$

$$\text{head}_i = \text{Attention}(\mathbf{X}W_Q^i, \mathbf{X}W_K^i, \mathbf{X}W_V^i).$$

In general, multi-headed attention provides a more flexible approach than single attention, as it allows the model to focus on different parts of the input sequence for each output element and obtain different types of information from the input.

Masked multi-head attention.

In the decoder, in the process of writing the text, future tokens cannot be taken into account, since they do not yet exist. This is called the auto-regression property - the ability to catch only the previous context.

Therefore, for each token it is crucial to mask the future part of the tokens during training so that the model learns to predict them based on the previous context:

To do this, a lower triangular matrix M of size $N \times N$ this format:

$$\mathbf{M} = \begin{bmatrix} 0 & -\infty & -\infty & \ldots & -\infty \\ 0 & 0 & -\infty & \ldots & -\infty \\ 0 & 0 & 0 & \ldots & -\infty \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \ldots & 0 \end{bmatrix},$$

is adding to compute the attention

$$\text{Mask Attention} = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}} + \mathbf{M}\right)V$$

$$\text{Mask Attention} \in \mathbb{R}^{N \times d_k}$$

Feed-Forward network. The goal of the FFN layer in the Transformer model is to apply a complex transformation to the embeddings of each token in sequence to capture more complex patterns and dependencies. While the attention mechanism considers the relationship between tokens in sequence, the feed-forward network learns information at the level of individual tokens.

FFN consists of two linear transformations with a ReLU activation function (formula in Appendix 4.1.1) between them. Formally, for input embeddings $\mathbf{X}$:

$$FFN(X) = \text{ReLU}(\mathbf{X}W_1 + b_1)W_2 + b_2,$$

$$W_1 \in \mathbb{R}^{D \times d_{ff}}, b_1 \in \mathbb{R}^{d_{ff}},$$

$$W_2 \in \mathbb{R}^{d_{ff} \times D}, b_2 \in \mathbb{R}^{D},$$

where $d_{ff} = 4D$.

Add and Norm.

Residual connection (**Add**) is adding the input value of embeddings to those obtained after transformation, for example, after multihead attention or running through a direct network. Formally:

$$\mathbf{X} = F(\mathbf{X}) + \mathbf{X},$$

where $F(\mathbf{X})$ are some transformations of the input embeddings. Residual connections help solve the fading gradient problem by providing a direct path for gradients to flow back to earlier layers.

The layer normalization (**Norm**) operation calculates the mean and variance of the input data over the last dimensionality (D) using the formula:

$$LN(\mathbf{X}) = \frac{\mathbf{X} - \mu}{\sigma}$$

where μ is the mean and σ is the standard deviation. The layer then applies two trainable parameters to scale g and to shift b the normalized data:

$$\mathbf{X} = g \cdot LN(\mathbf{X}) + b, g, b \in \mathbb{R}^{N \times D}.$$

Layer normalization is chosen over batch normalization because batch size is often 1 in language modeling.

Decoder-encoder attention.

In the decoder block there is a decoder-encoder attention, connecting the information received sequentially by the decoder about the current generated token in the context of the previous ones and the information received from the processing by the encoder about the input (source). The meaning of this block will be discussed in more detail in part 1.5.

1.1.2 Representation vs generation models

After the release of the architecture transformer, models began to appear that independently solve two problems:

- **Representation** - these models began to use Transformer blocks in their encoder structure to improve the extraction of features from the input information. Due to the mechanism of attention, it was possible to use the obtained embeddings to solve the problem of text understanding, semantic classification, etc. An example is the BERT model, whose architecture is discussed in part 1.4.1.
- **Generation** - models based on the Transformers decoder block began to develop in the tasks of generating texts and, due to attention, the model learned to predict the next tokens well, surprising such models as GPT, ChatGPT and others. The architecture of such models is discussed in part 1.2.

Architecture of models that independently solved the problems of representation or generation is described next, in order to be able to deeply understand the subsequent general solution.

1.2 Poetry generation

In my previous coursework ”Generating poetry using NLP tools”, the task was to generate poetry in the style of 5 famous Russian poets. The key idea for solving this problem was the idea of using the Russian model GPT-3 based on GPT-2 for fine-tuning on the corpus of Russian poetry. A separate model was used to train each author, because the small size of the model did not allow it to take into account the styles of all authors well at the same time, which is shown in part 2.3.1.

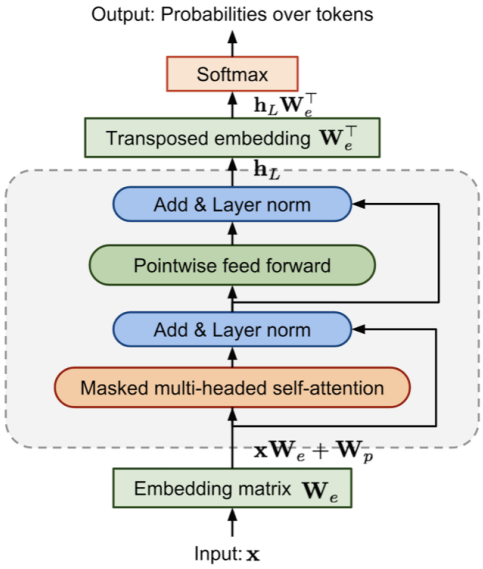


Figure 4: GPT-3 model architecture.

The GPT-3 architecture of the model, shown in **Figure 4**, looks almost like a decoder block, excluding the decoder-encoder, since the encoder is not provided in the model.

Basic moments:

1. To tokenize the model, the Byte-Pair Encoding (BPE) approach is used, which consists in the fact that each word is divided into frequently repeated characters of small length and one word can consist of several tokens.

2. Then each of these tokens (subwords) is replaced by the corresponding vector from the learned embedding matrix to form matrix $\mathbf{W}_e \in \mathbb{R}^{V \times D}$, where V is length of vocabulary.

3. To input embeddings of sentence $\mathbf{xW}_e$ adding learned positional embeddings $\mathbf{W}_p$

4. The number of repetitions of decoder block data is 24 for the average model.

5. At the final stage the resulting embeddings stored in the $\mathbf{h}_L$ state must be transposed using $\mathbf{W}_e^T$ to map the high-dimensional output vectors back down to the size of the vocabulary.

6. Then the softmax function applied to obtain the probabilities for each next token.

1.3 VIT: Visual transformer

1.3.1 Model overview

Transformers became a big breakthrough in natural language processing tasks after the publication of the article: Attention is all you need (Vaswani et al. [2017]), while for solving computer vision problems, convolutional neural networks (CNN) remained the dominant architect. However, inspired by the success of transformers in NLP, scientists have tried to integrate the attention mechanism into CNN (Wang et al. [2018];Carion et al. [2020]), or tried to replace CNN, but this was only effective in theory.

The direction of development of visual models was changed by a visual transformer (ViT), which in its architecture has few modifications compared to the original version. The general architecture of the model can be seen in **Figure 5**.

The problem of getting input embeddings. Original transformers receive a 1d vector of tokens of length N as input and then receive a two-dimensional matrix of embeddings. Unlike proposals, images have a 2D dimension, which means that images need to be adapted for presentation in 1D space.

As part of the solution, it would be possible to use each pixel - as a separate token, but this requires large computational costs. Transformers calculate mutual self-attention for each token with all other tokens in the sentence. If the sentence length is 256 characters, it is counting 256^2 attentions on one layer. Whereas the image

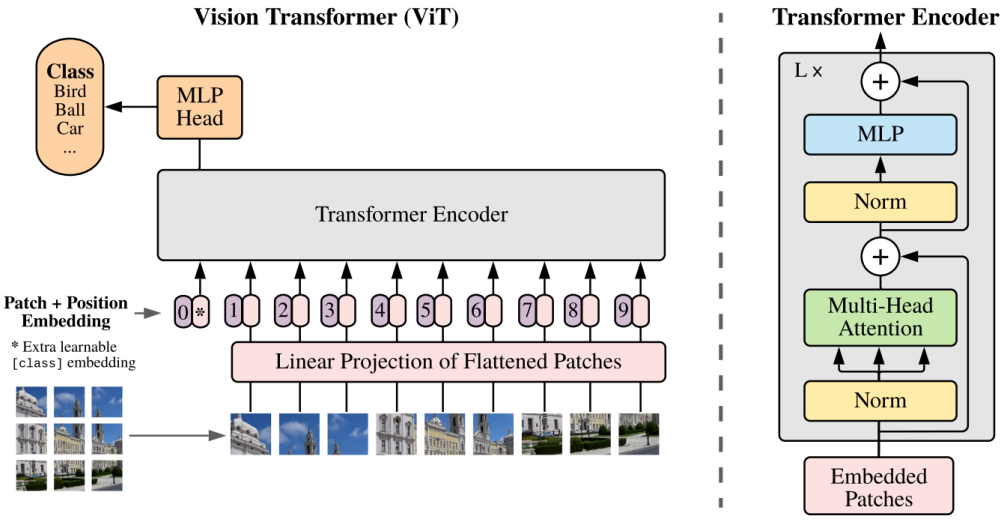


Figure 5: Vision Transformer (ViT) overview. Sourse: Dosovitskiy et al. [2020]

can be $256 * 256$ pixels and then it is consuming 256^4 attentions on each layer. The transformer also uses multi-head attention, that is, the number of calculations is multiplied by the number of heads.

The concept of localization of attention has appeared. To handle 2D images, the image is split into patches of a fixed size:

$$\mathbf{x} \in \mathbb{R}^{H \times W \times C} => \mathbf{x}_p^i \in R^{P \times P \times C}, i = 1 \ldots N$$

where (H, W) is the resolution of the original image, C is the number of channels, (P,P) is resolution of the patch and $N = \frac{HW}{P^2}$ is a number of patches. Then image patches are reshaping each patch into 1D array:

$$\mathbf{x} \in \mathbb{R}^{P \times P \times C} => \mathbf{x}_p^i \in \mathbb{R}^{1 \times P^2 C}, i = 1 \ldots N.$$

After that, each patch is multiplied by an initialized dense embedding matrix to obtain the input embeddings for the encoder, which will be denoted as z_1^i:

$$\mathbf{x}_p^i \in \mathbb{R}^{1 \times P^2 C} \cdot \mathbf{E} \in \mathbb{R}^{P^2 C \times D} = \mathbf{x}_p^i \mathbf{E} = \mathbf{z}_0^i \in \mathbb{R}^{1 \times D}, i = 1 \ldots N. \tag{3}$$

Or for whole image:

$$\mathbf{x}_p^i \in \mathbb{R}^{N \times P^2 C} \cdot \mathbf{E} \in \mathbb{R}^{P^2 C \times D} = \mathbf{x}\mathbf{E} = \mathbf{z}_0 \in \mathbb{R}^{N \times D}, i = 1 \ldots N, \tag{4}$$

where E is the learnable embedding matrix of the model, with shape $(P^2 * C) \times D$. The (3) formula is patch linear projection and it used to transform the flattened patches into the model's input representation, like it can be seen in **Figure 6**.

Positional embeddings. Adding the positional embeddings is still crucial. The authors' research in **Figure 7** has shown that trainable 1D embeddings give a relatively

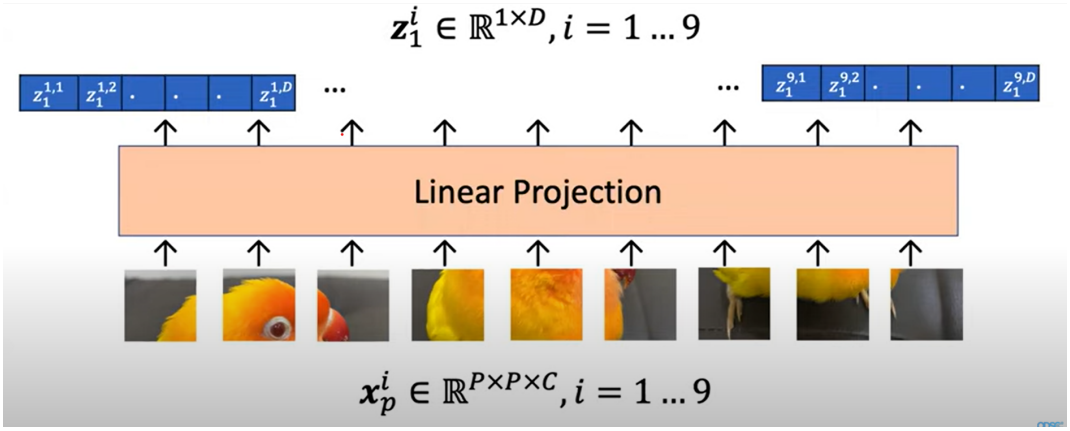


Figure 6: How patch linear projection works intuitively. Link to source: []

close result to trainable 2D embeddings. In their experiments, they used different approaches to the integration of positional embeddings, and also conducted experiments without positional embeddings and with relative rather than absolute positioning:

1. add at the moment of formation of input embeddings, after a linear projection;
2. adding a different set of learned positional embeddings at the beginning of each Transformer layer;
3. a single set of positional embeddings is learned and this same set is added to the input sequence at the beginning of each layer.

Pos. Emb.	Default/Stem	Every Layer	Every Layer-Shared
No Pos. Emb.	0.61382	N/A	N/A
1-D Pos. Emb.	0.64206	0.63964	0.64292
2-D Pos. Emb.	0.64001	0.64046	0.64022
Rel. Pos. Emb.	0.64032	N/A	N/A

Figure 7: Results of the ablation study on positional embeddings with ViT-B/16 model evaluated on ImageNet 5-shot linear (see in Appendix.4.2). Source: Dosovitskiy et al. [2020]

The table shows a big difference between the absence of positional embeddings and their presence, however, 1D and 2D do not have such a gap in different approaches, so it relevant to use the standard approach for transformers: initialization of positional embeddings in time of forming input embeddings in the one-dimensional case.

Sentence token. Another modification is that a trainable sentence token is added to the embeddings, initialized in $\mathbf{z}_0^0 = \mathbf{x}_{class}$ (similar to the class token in the BERT

model) and returned after the full encoder process in z_L^0 where L is a number of repetition of encoder block. This token can later be used, for instance, in the classification problem.

Accordingly, at the stage of patch embedding formation, which can be seen on the general transformer model on the right, the matrix is calculated:

$$\mathbf{z}_0 = [x_{class}; \mathbf{x}_p^1\mathbf{E}; \mathbf{x}_p^2\mathbf{E}; \dots \mathbf{x}_p^N\mathbf{E}] + \mathbf{E}_{pos},$$

$$\mathbf{E} \in \mathbb{R}^{P^2C \times D}, \mathbf{E}_{pos} \in \mathbb{R}^{N+1 \times D}.$$

Encoder. For this model, a transformer block encoder is used, repeated L times and one of the significant modifications from the original article about transformers: the integration of a multi-layer perceptron (MLP) as feed-forward network, i.e. a network that analyzes the information received about the mutual attention of tokens and draws conclusions. The MLP consists of two linear (or fully connected) layers, between which there is a non-linear GELU activation function, the formula of which is given in Appendix 4.1.2.

The MLP integration allows to add more abstraction to the model: while this part processes each position independently, and self-attention is considering the most semantically close parts at their own context.

It can also be seen that the output encoder of the model has an MLP head, which differs in the model for the state of pre-training and fine tuning. During pre-training, this MLP head consists of an input layer, a hidden layer, and an output layer, while during fine tuning it consists of only one linear layer.

Also Layernorm (LN) is applied before Multi-Head Attention and MLP blocks, and residual connections after every block, so the final process can be written as:

$$\mathbf{z}_l' = \text{Multi-Head}(\text{LN}(\mathbf{z}_l - 1)) + \mathbf{z}_l - 1, l = 1 \dots L$$

$$\mathbf{z}_l = \text{MLP}(\text{LN}(\mathbf{z}_l')) + \mathbf{z}_l', l = 1 \dots L$$

$$y = LN(\mathbf{z}_L^0),$$

where y is an image representation.

MLP classification head.

In the original article, a classification problem was used to train and evaluate the model, so the following solution was proposed for the problem.

The model, at the stage of running token embeddings (image patches) through an encoder block with repetitions, receives a rich representation of embeddings that reflects the essence of each input token, where the model learns to fill these tokens with functions useful for the current task. However, the ultimate goal is not to get embeddings, but to form a class label.

Therefore, the trained class token (the representation of the image by the model) is taken and run through the MLP head to get the class prediction.

The MLP head is different in the model for the pre-learning and fine-tuning state. During pre-training, this MLP head consists of an input layer, a hidden layer, and an output layer, while during fine tuning it consists of only one linear layer.

1.3.2 Breakthrough with CNN

Inductive biases.

In machine learning, there is the concept of inductive bias. The bottom line is that models should not remember one-to-one input information, but generalize the training data to unseen data and produce the corresponding output values. In fact, an inductive bias is a set of assumptions or a model's bias towards certain kinds of functions or predictions in order to generalize the processed data in order to be able to generate output values based on new data.

Different machine learning algorithms have different inductive biases. For example:

- Linear regression has a bias towards a linear relationship between input and output.
- Decision trees tend to be biased towards decisions that split data into pure subsets based on certain features.
- Neural networks have a more complex inductive bias, determined by aspects such as their architecture and the activation functions they use.

The effectiveness of inductive bias depends on whether the modeled assumptions match the true patterns underlying the data. If inductive bias matches these patterns well, it can allow a model to learn efficiently from a small amount of data.

Main features of inductive bias in convolutional networks:

1. Locality;
2. Two-dimensional neighborhood structure;
3. Equivariance of translation.

The convolution operation is applied to small local areas of the 2D input image, and each neuron in the hidden layer references previous layers that processed locally adjacent pixels. Thus, CNNs assume that neighboring pixels are more relevant to each other than distant ones. The weight distribution mechanism applies the same filters to the entire image, which makes it possible to find the same object in different parts of the image (equivariance).

The vit model has a smaller inductive bias, since its architecture contains attention mechanisms that do not have locality and translational equivalence. However, the layers of a multi-layer perceptron (MLP) are local and translationally equivariant. Due to the need to integrate positional embeddings, since the attention mechanism is computed for all tokens at the same time, the ViT model must learn all spatial relationships between tokens (patches) from scratch during training, unlike CNNs, which already have spatial relationships built into their architecture . Therefore, two-dimensional neighborhoods are used more carefully in this model.

Thus, the Vision Transformer model has less image-related inductive bias compared to a CNN. It has no strong built-in assumptions about image properties such as locality, translational equivalence, or two-dimensional neighborhood structure. As a result, the ViT model must learn these relationships from the data during training.

	Ours-JFT (ViT-H/14)	Ours-JFT (ViT-L/16)	Ours-I21k (ViT-L/16)	BiT-L (ResNet152x4)	Noisy Student (EfficientNet-L2)
ImageNet	**88.55** $\pm$ 0.04	87.76 $\pm$ 0.03	85.30 $\pm$ 0.02	87.54 $\pm$ 0.02	88.4/88.5*
ImageNet ReaL	**90.72** $\pm$ 0.05	90.54 $\pm$ 0.03	88.62 $\pm$ 0.05	90.54	90.55
CIFAR-10	**99.50** $\pm$ 0.06	99.42 $\pm$ 0.03	99.15 $\pm$ 0.03	99.37 $\pm$ 0.06	–
CIFAR-100	**94.55** $\pm$ 0.04	93.90 $\pm$ 0.05	93.25 $\pm$ 0.05	93.51 $\pm$ 0.08	–
Oxford-IIIT Pets	**97.56** $\pm$ 0.03	97.32 $\pm$ 0.11	94.67 $\pm$ 0.15	96.62 $\pm$ 0.23	–
Oxford Flowers-102	99.68 $\pm$ 0.02	**99.74** $\pm$ 0.00	99.61 $\pm$ 0.02	99.63 $\pm$ 0.03	–
VTAB (19 tasks)	**77.63** $\pm$ 0.23	76.28 $\pm$ 0.46	72.72 $\pm$ 0.21	76.29 $\pm$ 1.70	–
TPUv3-core-days	2.5k	0.68k	0.23k	9.9k	12.3k

Table 1: Comparison with the latest advances in popular image classification tests. Vision Transformer models pre-trained on the JFT-300M dataset outperform basic ResNet-based models in every way, while the preliminary preparation requires significantly less computing resources. ViT is pre-trained on the smaller public ImageNet-21k dataset also works well. Source: Dosovitskiy et al. [2020]

Less inductive bias makes the model train better and faster on a small dataset, as evidenced by research. The article about Vit models mentions that they trained ViT-H/14 (632M parameters and 14×14 patch size), ViT-L/16 (307M parameters and 16×16 patch size), as well as state-of- the-art CNNs such as Big Transfer (BiT) (Kolesnikov et al. [2019]) which is a supervised transfer learning with large ResNets and Noisy Student (Xie et al. [2019]), which is a large EfficientNet. **Table 1** shows the result of averaged mean and averaged standard deviation of the accuracy of pre-trained models on large datasets, obtained by fine-tuning (3 epochs) on small datasets, and the time spent on training.

From the key findings:

1. ViT-H/14 beats the best technology while learning 4 times faster;
2. ViT-L/16 trained on a much smaller dataset gives results only a few points worse than ViT models trained on a huge dataset.

Conclusion: this proves the optimality of fine-tuning not the largest pre-trained

model based on transformer architecture for solving a specific problem and the lack of advantages in developing a model only for this specific problem.

CNN instead of linear projection. The approach of patching and using them directly to generate input data to an encoder is considered a pure or raw solution for the architecture. Instead, it is possible to use a CNN to extract the feature map.

The following changes are made to the previous approach:

1. An image of size $H \times W \times C$ is fed into CNN to process the image and output a feature map of size $H' \times W' \times C'$.

2. Feature map is divided into N patches each of size $P \times P \times C'$. Here C' is the number of channels in the feature map, which is usually much larger than C as it represents more complex features.

$$\textbf{features} \in \mathbb{R}^{H' \times W' \times C'} => \textbf{features}_p^i \in R^{P \times P \times C'}, i = 1 \dots N,$$

   where $N = H'W'/P^2$

3. Each patch is shaped just like it was in the formula (3)

4. Then each patch is multiplied by the dimension embedding matrix in the same way as in equation (4)

In the special case where the patches are 1x1 in size, step 2 simply reduces the feature map to a sequence, and step 3 is the same linear transformation applied to each unit of the feature map.

Therefore, the hybrid approach combines the power of CNNs in extracting local features with the ability of Transformers to model long-term dependencies.

Consequently, in this part of this chapter, the technology for extracting information or features from an image was considered in detail. The hybrid model was used to solve the problem of generating poems and music from images, the architecture of the finished solution and the integration of the ViT model will be given in part 1.5.

1.4 Keywords extractors

As part of one of the solutions, the technology of sequential generation from an image to keywords and from keywords to poetry was used and tested.

To do this, it was necessary to explore the possibility of extracting keywords from the text. In this part, it was considered the keyword generation approach based on the Sentence-BERT model, which was using with KeyBERT library.

1.4.1 Architecture of BERT model

Before explaining how keyword extraction using Sentence-BERT works, it is necessary to consider low-level base-model. Sentence-BERT is based on the language representation BERT model, which stands for Bidirectional Encoder Representations from Transformers, with some modifications.

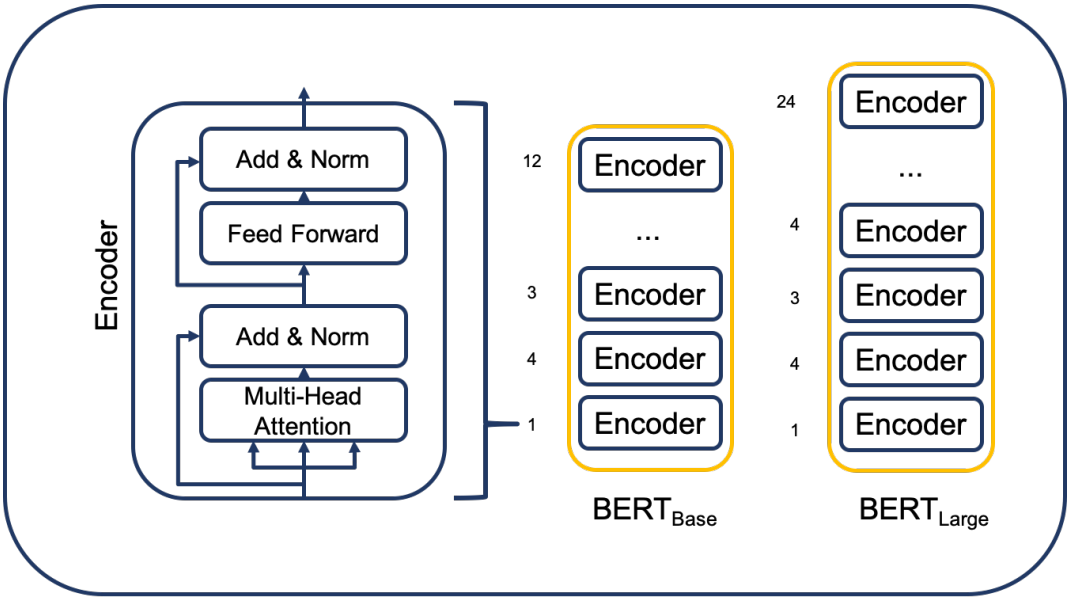


Figure 8: BERT model architecture.

Basically, the architecture of the model consists of layers of the encoder block type in Transformers, and depending on the configuration, in the process of pre-training the model, the layers were repeated either 12 (basic) or 24 (large) times, which can be seen in **Figure 8**.

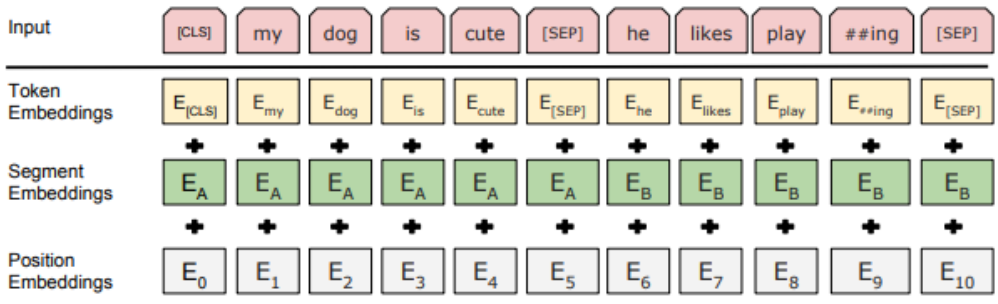


Figure 9: BERT input embeddings. Sourse: Devlin et al. [2019]

Two sentences are fed to the input of the bert model, with the addition of a classification token - a token that, in the process of pretraining, will form a representation of the entire sentence, and a token for dividing two sentences. As shown in the **Figure 9**, **input embeddings** are formed from:

1. **token embeddings**, which are initialized randomly for pre-training in order to train them;
2. **positioning embeddings**, which are needed for the attention mechanism;
3. **segmentation embeddings**, which are needed to understand the model during mutual attention calculation which embeddings are responsible for sentence A and sentence B.

The contextual embeddings of each word returns as the output value of each layer of the encoder, and accordingly at the output of the whole model.

How were contextual embeddings trained?

Traditional language models use sequential prediction of the next word, that is, the connection between the encoder-decoder layers of blocks goes from left to right. In the case of the burt model, two-way communication between layers was used to obtain a two-way contest representation of each word, that is, each current token took information from previous layers about its right and left neighbor.

Masked Language Modeling

In the pre-training process, 2 approaches were used for training: This is how BERT learns the contextual relationship between words. 15% of all tokens in the sentence were randomly masked and the model had to return the embeddings of the masked word.

Input sentence was run through the Transformer layers and the resulting embeddings are transferred to a fully connected (dense) layer with softmax activation, which returns the probability distribution over the dictionary. The word with the highest probability is selected as the prediction for the masked word.

For training, it was necessary to evaluate the quality of the returned embedding for the masked word and minimize the error, so the loss function was calculated using the cross-entropy formula:

$$H(p,q) = -\sum_{i=1}^{V} p(i)\log q(i) = p(\text{true word})\log q(\text{predicted word}),$$

where i is consecutively all words in vocabulary V, $p(i)$ is a probability for ith-word to be instead of [MASK] token, and $q(i)$-predicted probability of word, selected as the prediction for the masked word. In this case $p(i)$ for each true masked word is 1 and others are 0, so sum is not necessary.

Next Sentence Prediction (NSP) To understand the relationship between sentences, BERT is trained with a task called Next Sentence Prediction.

For each training example, two sentences are selected from the corpus used for training. In 50% of cases, the second sentence is chosen as the one that follows

the first one in the source document (correct pair), and in the other 50% of cases, the second sentence is a random sentence from the training corpus (false pair).

The output corresponding to the [CLS] token is passed to the simple binary classification layer. The purpose of this level is to predict whether the second sentence follows the first sentence in the source document or not, and return 1 or 0. This prediction is compared to the true label (1 if the second sentence actually follows the first sentence, and 0 otherwise), and the loss is calculated. This loss is then used to update the model parameters through error backpropagation.

1.4.2 Architecture of Sentence-BERT

The Sentence-BERT model uses the BERT model but gets rid of the final classification head and processes one sentence at a time. It adds a pooling layer to the returned embeddings to get a single vector from the embedding matrix. In this model pooling represents the Mean Pooling, so for sentence with N tokens and the token embeddings are denoted as $E_1, E_2, \ldots, E_N$, operation is performed as follows:

$$\text{Sentence Embedding} = \frac{1}{N}\sum_{i=1}^{N} E_i$$

Unlike BERT, SBERT is fine-tuned to sentence pairs using a Siamese architecture, i.e. two identical BERTs are used in parallel that have the same network weights.

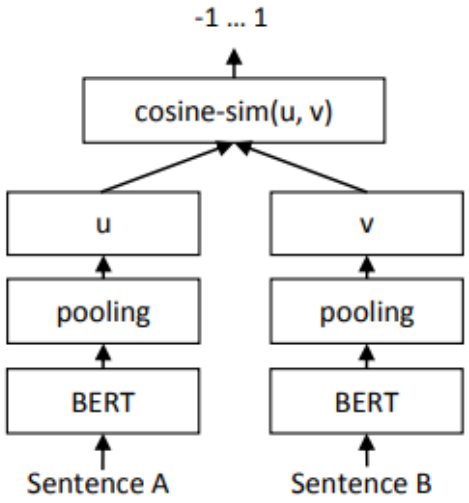


Figure 10: Sentence-BERT inference stage architecture. Sourse: Reimers and Gurevych [2019]

As shown in **Figure 10** during inference, the model takes the resulting vectors from the two compared sentences and calculates the cosine similarity, returning the resulting value.

1.4.3 KeyBERT process

Having considered the architecture of the model Sentence-BERT, built on the basis of the architecture of the BERT model, the general pipeline of the work of the library KeyBERT is as follows:

1. From the input sentence, n-grams (candidates for keywords/sequences) are extracted, that is, sequences of words of length n, and words can only be adjacent. The parameter n is set manually.
2. Using the Sentence-BERT model, the cosine similarity between the proposal and each candidate is estimated.
3. Among the candidates, the top-k sequences with the greatest cosine similarity to the original sentence are selected. The closer the cosine similarity to 1, the better the candidate captures the meaning of the original sentence.

1.4.4 Other Methodologies

As part of the study, other approaches were also considered for obtaining a short textual formulation from poetry. For example, models based on transformers were used, which paraphrase or summarize the text, however, with the Russian language, and especially with specific Russian poetry, these models coped rather poorly: they returned either some kind of cut sentence, or a piece of words that did not reflect the essence. More details in part 2.1.2. Technologies for extracting keywords were also considered, for example, with a deterministic approach, such as RAKE. But he also worked poorly in Russian poetry, since it contains a large number of rare and specific words.

Therefore, the SBERT model was used as the final solution in this problem, since the results produced corresponded to the requirements, which can be seen in more detail in part 2.3.1.

1.5 Cross-attention

Above, various models are considered that have, within their architecture, only an encoder or only a decoder blocks of transformers. What is interesting is that from such models: representative and generative, it is possible to make a general encoder-decoder model capable of performing a solution at the intersection of sciences. For example, use a visual encoder to extract information from an image and a text decoder to generate text based on the information received.

It can be seen that in the architecture of the generative model of the GPT type (**Figure 4**), one type of layers is missing, which were given in the original Transformers article (**Figure 2**). This block is called **cross-attention**, since attention is

calculated here not only by the input information from previous decoder steps, but also by the output information received from the encoder.

Cross-attention is calculated in the same way as presented in the formula (1), but with some modifications. Query is formed based on decoder embeddings, Value and Key is computed based on encoder embeddings. Formally, if $\mathbf{X}_e$ is encoder embeddings and $\mathbf{X}_d$ is decoder embeddings:

$$\text{Cross Multi-head Attention} = \textbf{softmax}(\frac{(W_Q^i \mathbf{X}_d)(W_K^i \mathbf{X}_e)^\intercal}{\sqrt{d_k}}) W_V^i \mathbf{X}_e$$

$$\mathbf{X}_d, \mathbf{X}_e \in \mathbb{R}^{N \times D}, i = 1, \ldots, h$$

And the (2) formula then calculates the multi-head attention for h heads.

If it is necessary to build a model containing a visual encoder and a text decoder, then the combining these models into one common transformer is needed. Looking at the independent architectures of a ViT model that contains an encoder block and is able to extract features from an image (**Figure 5**), and a text model like GPT-3 that is able to decode input embeddings into a sentence and consists only of a decoder block (**Figure 4**), addings some elements to get a full-fledged Visual Encoder Decoder model, will be relevant.

1. Initialize the weights and the entire block of cross-attention. For using pre-trained models that are good for simpler tasks, it is necessary to initialize the cross-attention weights and finish training them on the picture-text corpus.

2. Instead of the last MLP of the classification head, use only a line layer in order to convert the dimension of the embeddings received from the encoder into the dimension of the embeddings required by the decoder.

As a result of such a combination of a visual encoder and a text decoder, an updated visual-text transformer is obtained, the architecture of which is visible in **Figure 11**.

1.6 Text2music generation: BART

This part discussed technologies that are applicable to solve the individual problem of generating music based on text. The task of generating music itself looks complex, since there are several types of music representation, for example, in the form of a spectrogram, which is great for retraining with diffusion models. Within the framework of this work, the task of generating music will be considered as a task of natural language processing, since this became possible due to the existence of ABC notation.

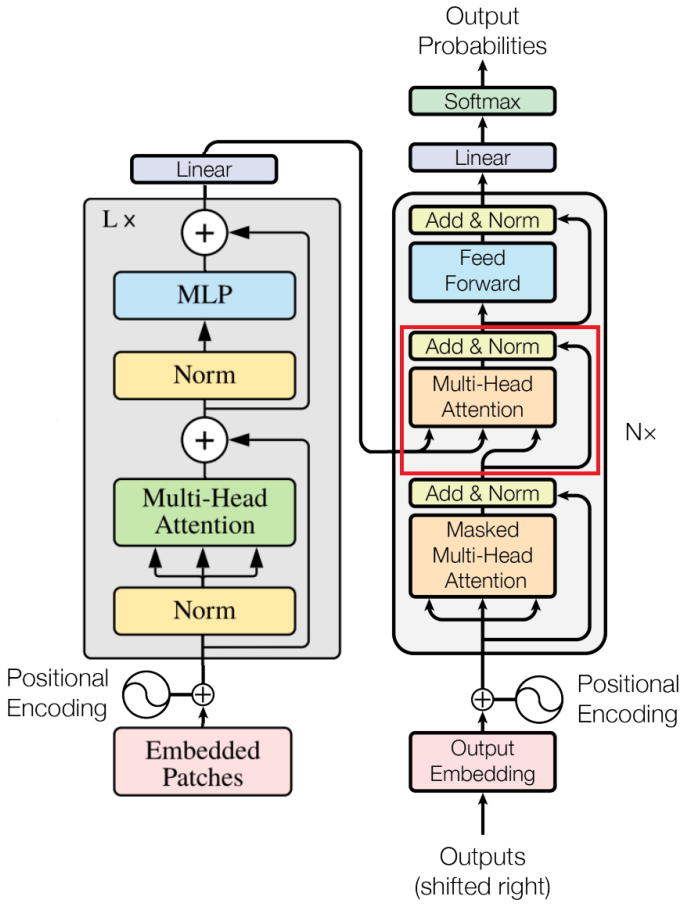


Figure 11: Combination of ViT model (vision transformer with encoder block:**left**) and GPT-3 model (text transformer with decoder block:**right**), adding cross-attention block (red) and linear projection from encoder to decoder.

1.6.1 ABC notation

ABC notation is an abbreviated form of musical notation based on text. It was originally designed to record tunes of Western European origin (such as Irish, English and Scottish folk music), but it can also be used to record music from a variety of other traditions.

In ABC notation, music is written using letters, numbers, and symbols that are easy to type on the keyboard. This makes ABC particularly suitable for exchanging music notation online or as text files.

The structure of the ABC notation is quite simple:

- Title: Each piece of music begins with a title, which includes various fields to describe the piece. This includes the name (T:), composer (C:), meter (M:), key (K:), and default note length (L:). Each of these fields starts with a letter

followed by a colon and then the relevant information.

- Main part: The main part of the work is the place where the real music is recorded. Notes are denoted by the letters a-g (or A-G for the lower octave). Note duration is indicated by numbers after the letter (for example, A2 for a note twice as long as the default, A/2 for a note half as long).
- Additional Symbols: Various other symbols are used to represent other musical symbols, such as "∧" for sharp, "=" for natural, and "_" for flat. Residues are marked with a "z". Repeats are denoted by the symbols "|:" and ":|".

1.6.2 Data

Recently, an article was published on the use of pre-trained generative models for solving the problem of generating music based on text Wu and Sun [2023], where all models were trained on a huge closed TextTune dataset. The dataset consists of 282,870 English texttune pairs, in which the text description is matched to the music in ABS notation. Text descriptions are quite limited in creativity as they only include:

1. analysis of tonality, harmony, musical style;
2. musical terms about the concepts of rhythm, speed, mood of music;
3. historical background;
4. sensory analysis.

The ABC notation allows you to store music as a sequence of symbols that can be decoded by special libraries. Accordingly, the task of generating music is then reduced to the task of generating a specific text, taking into account its rules. The dataset contains various genres that allow you to add variety to the generated music. Executed scores are written on one note line, which greatly simplifies the task. Scores have a minimum of 8 measures to ensure the completeness of the musical thought.

1.6.3 Evaluation

For the study, different models were used, the configuration of which is presented in **Table 2**. It is also interesting that the GPT-2 model, which is a decoder, was considered in the article as an encoder. BART breakpoints initialize the encoder and decoder, and other checkpoints only initialize encoder. Number of parameters in encoder/decoder breakpoints are slightly different due to different vocabulary size.

RND is a randomly initialized encoder using WPT for tokenization (like GPT-2), with a token frequency limit of at least 100.

The quality assessment of pretrained models is given for different metrics:

Checkpoint	Layers	Hidden	Heads	Params
RND	12	768	12	91M
BERT	12	768	12	109M
GPT-2	12	768	12	117M
BART-base	6+6	768	16	139M
BART-large	12+12	1024	16	406M

Table 2: The configurations of pre-trained models for fine-tuning on text2music dataset. Source: Wu and Sun [2023]

- BLEU-N: Measures the proportion of n-gram sequences in the source text that are also present in the generated text. First presented in the article Papineni et al. [2002].
- DIST-N: Assesses the diversity of generated examples based on the proportion of diverse n-gram sequences. First presented in the article Li et al. [2016].
- EDS (Edit distance similarity): Metric based on Levenshtein distance to calculate the similarity of the real and generated version, normalized for the range from 0 to 100, which computes as:

$$EDS(a,b) = \frac{(1 - lev(a,b))}{max(|a|,|b|)} \times 100,$$

where $|a|$ and $|b|$ are the length of two samples.

Checkpoint	BLEU-2	BLEU-3	BLEU-4	DIST-1	DIST-2	DIST-3	EDS
RND	44.47±20.64	35.88±18.12	28.84±16.22	**10.61±4.32**	**28.47±11.03**	**41.79±15.24**	38.02±13.10
BERT	21.72±14.01	15.44±10.61	10.47±7.90	8.71±5.49	19.03±11.95	27.09±17.03	24.81±10.40
GPT-2	46.76±21.23	38.20±19.41	31.14±18.14	10.39±4.39	27.86±11.28	40.93±15.70	39.94±14.84
BART-base	**48.34±21.47**	**39.85±20.11**	**32.82±19.22**	10.29±4.31	27.98±11.01	41.28±15.15	**40.77±15.75**
BART-large	22.41±14.28	15.96±10.71	10.91±7.78	9.21±6.36	20.54±13.28	29.09±18.04	24.98±10.27

Table 3: Results of pre-trained models fine-tuned for text-to-music generation on the validation set. Source: Wu and Sun [2023]

As can be seen in the **Table 3**, the BART model in the basic configuration gives better results compared to other models, and at the same time, it can be seen how close they are to the truth with GPT-2.

1.6.4 BART Architecture

Diving deeper into the architecture of the BART model, which proved to be the best at solving this problem, one can notice some interesting solutions.

For the first time this model was developed in the article Lewis et al. [2019] and the presented architecture of the model at **Figure 12**.

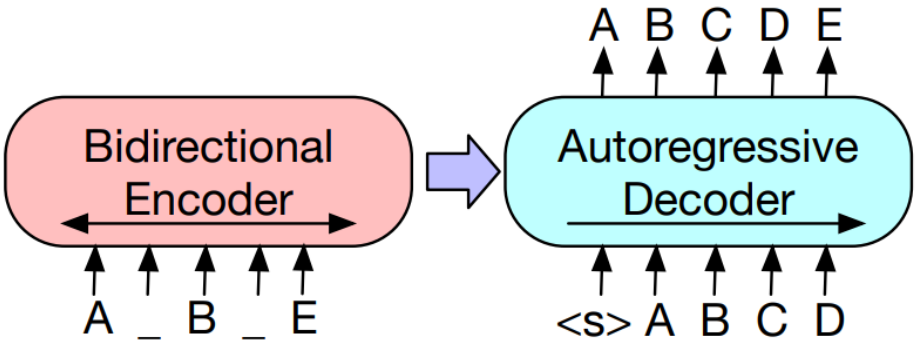


Figure 12: The overall BART architecture. Sourse: Lewis et al. [2019]

Instead of independently trying to use only the BERT model or only the GPT-2 model, a coupled version of them is used with the addition of cross-attention as in the original transformer structure.

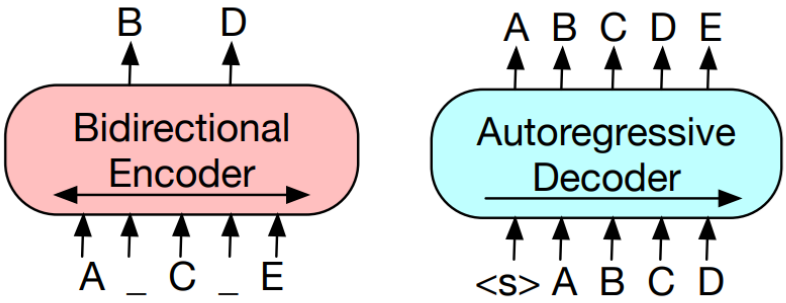


Figure 13: The BERT architecture **(left)** and the auto-regressive decoder (GPT-2) architecture **(right)**. Sourse: Lewis et al. [2019]

The BERT model (**Figure 13 left**) is not capable of generating text in human language and its advantage lies in the bidirectional communication between the encoder layers to extract the context. Random tokens are replaced with masks with masks and missing tokens are predicted independently.

The peculiarity of the GPT-2 (**Figure 13 right**) model is that it can generate autoregression (that is, sequentially) text, but the connection between layers from left to right, so that only the previous context can be taken into account, not the future one.

The joint integration of these models allows you to use the advantages of a bidirectional encoder that detects a two-way context and an autoregressive decoder that can generate text sequentially based on the previous context. For instanse, BART

is pre-trained with specific input data corruption, like BERT masked 15% of tokens during training, to learn the model to reconstruct the text without corruptions.

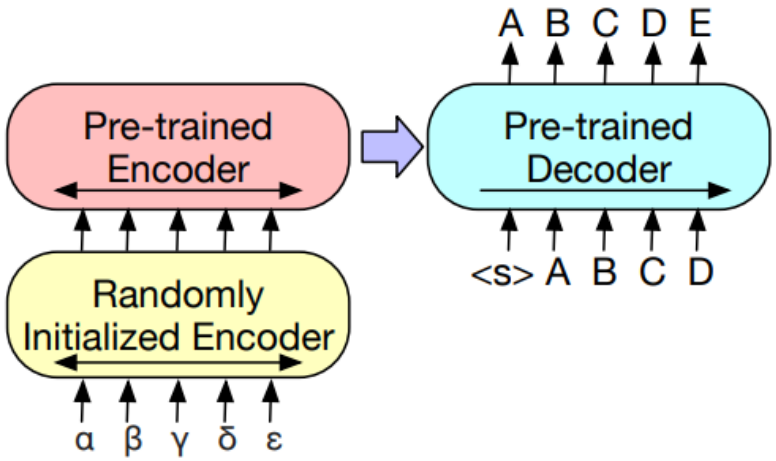


Figure 14: The BERT architecture **(left)** and the auto-regressive decoder (GPT-2) architecture **(right) for fune-tuning on more complicated task**. Sourse: Lewis et al. [2019]

The article also provides the architecture of using the Bart model for additional training for a more complex task, for example, the task of translating from another language. Then, as shown in the **Figure 14**, an additional encoder is added to the main model, initialized randomly, instead of the part of constructing the first words of the bidirectional encoder embeddings. The new encoder can use a disjoint vocabulary (the encoder is using a different vocabulary than the rest of the model). Further, this model is trained on the text corpus for translation in order to optimize the weights of the additional encoder.

1.7 CLIP Processor

CLIP (Contrastive Language - Image Pretraining) is a model developed by OpenA and presented in the article Radford et al. [2021] that connects images and text into a single, shared embedding space. The CLIP architecture is shown in **Figure 15** and consists of a ViT model for image processing and a model for language processing (like the BERT model).

The key innovation is to teach the model to understand which title corresponds to which image. This was achieved with a large dataset of 400 million pairs (image, text) collected from the internet.

The CLIP model is trained using the contrast target:

- During training, images and associated text descriptions are processed separately by vision and text transformers.

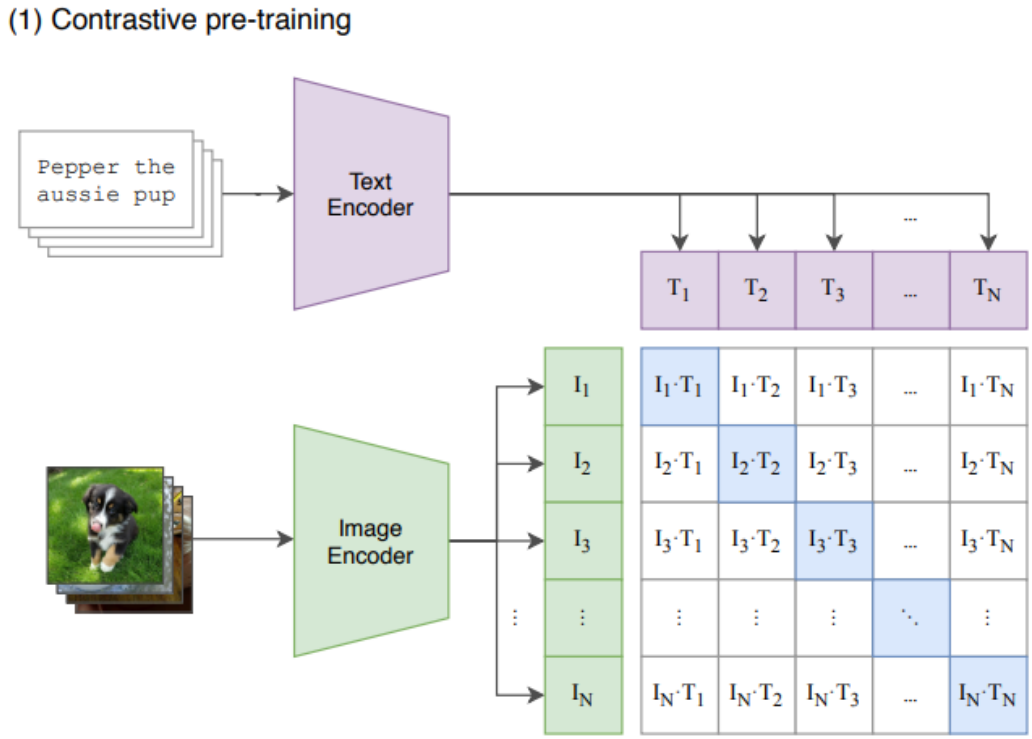


Figure 15: CLIP model architecture that trains visual and text encoder at the same time to better extract features from image-text pairs and find their relationship. Source: Radford et al. [2021]

- Then the model is trained to maximize the agreement (using cosine similarity) between each correct image-text pair and minimize the agreement between all other pairs.

Let's denote the images and text representations as **V** and **T**, respectively. For each i-th image there is an i-th text pair.

The similarity between image i and text j is calculated as the scaled dot product of their representations, which can also be interpreted as cosine similarity if the representations are normalized.

$$sim(i,j) = \mathbf{V}_i^T \mathbf{T}_j,$$

The point of the pair (i,i) is to maximize the utilization while minimizing the remaining pairs (i,j) and (j,i) for all $j \neq i$. This can be stated as the minimization of the following contrast loss function, also known as noise contrast estimate (NCE) loss:

$$L_i = -\log\left(\frac{\exp\left(\frac{\text{sim}(i,i)}{\tau}\right)}{\sum_{j=1,j\neq i}^{N}\exp\left(\frac{\text{sim}(i,j)}{\tau}\right)}\right) - \log\left(\frac{\exp\left(\frac{\text{sim}(i,i)}{\tau}\right)}{\sum_{j=1,j\neq i}^{N}\exp\left(\frac{\text{sim}(j,i)}{\tau}\right)}\right),$$

where τ is a temperature parameter and parts inside log is softmax formula.

The total loss is the sum of L_i over all i instances. The model is trained by minimizing this loss function. By learning in this way, the model learns to map images and their associated text to adjacent vectors in a common embedding space.

This powerful combination of visual and text encoders allows CLIP to use learned associations between images and text to solve other problems on new data using zero-shot learning, which means that it can generalize its learning to solve problems that it did not see during training. For example, CLIP can perform the task of identifying an image from a set of previously invisible categories simply by using their descriptions.

In conclusion, in this chapter, a theoretical basis was given for the use of technologies for **encoder-decoder models** based on transformers, including independent solutions for visual (**ViT model**) or textual (**BERT model**) extraction of features or features of input information. Approaches were also considered for generalizing the text, identifying keywords based on the received embeddings using **Sentence-BERT**, which will be further used to assess the quality of the trained models. Referring to the architecture of the model studied in the course work on the generation of poems (**GPT-3)**, the possibility of using pre-trained independent encoder and decoder models in conjunction with the use of **cross-attention** was considered from a theoretical point of view. As a justification for this possibility, a **BART** model was also introduced that is capable of performing the task of generating music from text. Also, the **CLIP processor**, more sharpened for the task of picture and text, was considered for use as an image processor for the encoder-decoder model being trained.

2 Poetry generation from images

This part discusses practical approaches for realizing the task of generating poetry from an image, including the evaluation of the use of different sizes of models and certain concepts. The whole project can be found at GitHub repository by link (https://github.com/anyaschenikova/image2poetry).

2.1 Image-keywords-poetry

One of the first and easiest ideas to implement was not a holistic training of one model to use the input image and get the final poetry, but an intermediate step-by-step generation that would avoid large data collection and search for resources.

The main task in the solution was to study the language model for the generation of verses from some generalization (keywords or one sumarizing sentence), but as described in part 1.4, as a result, extraction of keywords was used due to the more efficient technology.

2.1.1 Data preparation

As part of the coursework on the generation of Russian-language poetry in the style of famous authors, a dataset was formed in which each large block of poetry (the whole work) was divided into small pieces of 12 lines and the dataset was cleaned up for long unrhymed comments and extra characters.

However, within the framework of these small pieces, there was an understatement of thought, since 12 lines could begin from the middle of a logical sentence and end with an unfinished sentence. The disadvantage was also that despite the fact that 12 lines were allocated because such a number of lines should take into account all the variations of the rhyme, in reality this turned out to be wrong due to shifts and the continuation of the rhyme of one sentence could end up in another piece. Also in the dataset, a clear imbalance was revealed for each author, for Mayakovsky there were only 404 examples, for Blok 1900.

Accordingly, it was decided to reformat the dataset as follows:

1. Each large work was divided into sentences (a complete logical thought).
2. Sentences were cleaned for extra characters, but not specific to the author, extra spaces, foreign characters (French and English speech is excluded).
3. Sentences were sequentially combined into a piece until the length of the piece (the number of lines) did not exceed much 12 and at the same time its length was even. This comes from the assumption that each rhyme must occur at least 2 times and that the logical ending of a small rhyming block lies in one sentence or in a combination of several.

4. If one work consists of a small number of lines (less than 20), then it is not cut into smaller pieces. The minimum rhyming block is more than 4 lines.

5. The name of the poet in English was changed to Russian.

As a result, a more balanced dataset was assembled with logically completed rhyming blocks, in which the number of examples for each poet ranged from 1000 to 2000.

2.1.2 Keywords extraction

In order to train a model for generating poems from some kind of generalization, it was necessary to assemble a dataset of generalization-poetry pairs, and for this, the resulting poetry had to be run through some kind of technology to extract the generalization.

First, summation models were used, which were supposed to derive a short sentence reflecting the key idea based on the input text. Within the framework of the models under consideration, various advanced versions of the T5 model, a neural network model capable of understanding and generating text (text-to-text transfer transformer), were used. The main idea of the model is to use the encoder-decoder architecture from transformers in order to train it for various mini-tasks: translation, summation, etc. In the original version of the article Raffel et al. [2019], the authors released two models: a large one for solving 24 problems and a multilingual one (101 languages, mt5) for generating words in place of the mask.

There are several fine-tuning models for the Russian language in the public domain:

- **cointegrated/rut5-base-multitask** - only embeddings of tokens for Russian and English were extracted from the MT5 model and trained on the tasks of translation, summarization, paraphrasing, filling in gaps in the text, continuing the text.

- **IlyaGusev/rut5-base-headline-gen-telegram** - additional training of the previous model for headline generation

- **csebuetnlp/mT5-m2o-russian-crossSum** - The mT5 many-to-one (m2o) breakpoint is configured on all cross-language pairs of the CrossSum dataset where the target summary was in Russian, i.e. this model tries to summarize text written in any language in Russian.

- **cointegrated/rut5-base-absum** - This is a model for abstract generalization of the Russian language based on **cointegrated/rut5-base-multitask** and tuned to 4 datasets for summary.

As an example, to assess the quality of the summation, Mayakovsky's poem, shown in the **Figure 16** , was chosen, which is distinguished by the specificity of language

Один не смогу —
не снесу рояля
/тем более —
несгораемый шкаф/,
А если не шкаф,
не рояль,
то я ли
сердце снес бы, обратно взяв.
Банкиры знают:
«Богаты без края мы.
Карманов не хватит —
кладем в несгораемый».
Любовь
в тебя —
богатством в железо —
запрятал,
хожу
и радуюсь Крезом.
И разве,
если захочется очень,
улыбку возьму,
пол-улыбки
и мельче,
с другими кутя,
протрачу в полночи
рублей пятнадцать лирической мелочи.

Figure 16: Mayakovsky's poetry example for summary and keywords extraction evaluation.

constructions and heaps of meaning: many words showing shades of emotions and essence.

Different models either cut out a sentence or a piece from the example that reflected the essence as much as possible, or tried to generate their own (**csebuetnlp/mT5-m2o-russian-crossSum**), but a little missing the point. In any case, the output summarization did not give the generalization that is required for the task, because the user is unlikely to be able to give such n for prompts for writing a poem and it is very difficult to adapt the visual model for the image capture task to generate such prompts.

Therefore, a more general solution was needed, such as extracting keywords. The common approach described in part 1.4 was used, namely the use of the KeyBERT library with Sentence-BERT integration to extract embeddings. There are a large number of different models for the formation of embeddings, but I've considered multilingual ones in order to take into account the Russian language.

Model	Generated summary
cointegrated/rut5-base-multitask	Один не смогу — не снесу рояля
IlyaGusev/rut5-base-headline-gen-telegram	Любовь в тебя — богатством в железо — запрятал, хожу и радуюсь Крезом
csebuetnlp/mT5-m2o-russian-crossSum	Если бы я могла снести нечто такое, как будто в твоем сердце.
cointegrated/rut5-base-absum	Любовь в тебя — богатство в железо.

Table 4: Comparing generated summary according to different fune-tuned models.

Two models were used for evaluation and selection:

- **xlm-r-large-en-ko-nli-ststb**(default) in other words, based on RoBERTa's model, as shown in Listing 4.5. This model was used for keywords extraction in final version.
- **distiluse-base-multilingual-cased-v1** in other words, based on the DistilBERT model.

Both RoBERTa and DistilBERT are based on the architecture of the BERT model with the only difference: RoBERTa improve on the performance while DistilBERT improves on the inference speed.

Model	Generated keywords	Cosine similarity
xlm-r-large-en-ko-nli-ststb (default)	богатством	0.4097
	железо	0.1733
	захочется	0.166
	пятнадцать	0.0483
	один	0.0066
distiluse-base-multilingual-cased-v1	'шкаф'	0.2504
	'сердце'	0.2445
	'несгораемый'	0.1431
	'один'	0.0904
	'пятнадцать'	0.0285

Table 5: Comparing generated keywords according to different fune-tuned models. Cosine similarity reflects the value of the closeness of the keyword with the meaning of the entire sentence.

The result of using these models on the considered example of poetry is shown in the **Table 5**. For the first model, the result is more interesting and scattered, since a word was found that reflects the meaning of the example by 40%, while the second

model shows average results close to 10 – 15% for all keywords. Therefore, to form the final version of the dataset, the **xlm-r-large-en-ko-nli-ststb** model was chosen. Dataset can be found by link.

The code for generation keywords from text can be found at Appendix 4.6.1.

2.1.3 Fine-tuning the GPT-3 for generation from keywords

The main idea of training a model for generating poetry from keywords is to fine tune the model on a body of text, in which keywords come first, then poetry. At the stage of inference of the model, keywords are supplied to the input prompt, and at the output, the model supplements the input information with generated poetry.

As part of finetuning, all model weights are optimized, so the process requires both time and computational resources, but not as much as during training.

Автор: Маяковский
Ключевые слова: любовь, жизнь, сон
Поэзия: Жизнь на!
Дай поцелую!
Милая!
Нежная!
В сон!
В жизнь!
Поверь,
лучше нету!
Люблю твой стан упругий,
Твои шелковые космы,
И синие глаза,
И смех сквозь слезы
Твои,
Люблю,
Нежная,
Ириска моя!

Figure 17: Schema for input data for model finetuning to generate text from keywords in the style of a given author.

To train the model, the code given in 4.6.2 was used and was discussed in my coursework of generating poetry in more detail with the only difference: in the process of creating a dataset for training, instead of using only poetry, the input is formed according to the structure shown in **Figure 17**.

For fine-tuning for the task of generating poetry from keywords, the **ai-forever/rugpt3-large-based-on-gpt2** and **ai-forever/rugpt3-medium-based-on-gpt2** models were

used. The differences in these models are shown in **Table 6**.

Model	Parameters	Model size, GB	Number of GPUs used in training
RuGPT3-small	117M	0,5	32
RuGPT3-medium	345M	1,6	64
RuGPT3-large	762M	3,14	128

Table 6: Information about model: amount of parameters, size and number of GPU's needed for training process

.

Also, from the innovations, as part of the solution of the current work, the training was produced not separately for each author, but simultaneously on the entire data corpus with the author indicated. The evaluation is described in part 2.3.1.

2.1.4 Form a generation pipeline

In this case, the general generated pipeline will look like this, the code is given in 4.6.3:

1. Obtaining a model for solving the image-captioning problem.

   As part of such a pipeline, the **tuman/vit-rugpt2-image-captioning** model is used, which can generate a description in Russian from a picture. The overall architecture of this model can be found in part 1.5.

   Definition 1
   *The model **tuman/vit-rugpt2-image-captioning** is the **base model** for image to poetry or music task.*

   The main idea of this model was described in part 1.5: As an encoder, a pre-trained ViT model (architecture of which is explained in part 1.3) was used, and as a decoder, a pre-trained Russian model **ai-forever/rugpt3-large-based-on-gpt2** was used. This created visual encoder-decoder model was fune-tuned on the COCO image dataset (2014) Lin et al. [2015], only translated into Russian to resolve image captioning problem in Russian.

2. Removing punctuation and stop words and obtaining a normal form of words. Each remaining word was then independently considered as a keyword. This stage is needed to form keywords on the basis of which poetry will be generated.

3. Obtaining a model for generating poetry from keywords for the selected author. In this case, a large or medium model is used, fine tuned on input text like schema in **Figure 16**. It is necessary to choose a poet in whose style poetry will be generated, and the keywords obtained in the previous step will be used as the basis for generation.

Note that the decoder in model for solving the image captioning task, is identical in architecture to the model used to generate poetry from keywords. This laid the foundation for the following solutions to the problem.

2.2 Visual Encoder-Decoder model

In this part, approaches will be considered in detail for solving the problem of generating poetry from an image based on the creation of Visual Encoder-Decoder model, which includes a visual encoder and a text decoder.

2.2.1 Data collection using Stable Diffusion

First of all, to solve the problem of creating a full-fledged model for such a generation, it is necessary to collect a dataset, on the basis of which training will take place. The parts below will describe what exactly helps to train such a specific dataset.

To collect a dataset consisting of triplets: picture - author-poetry, it would take a large number of hours of manual search for the appropriate image for each piece of poetry.

To simplify task, it can be relevant to use the Stable Diffusion open source model, which can generate images well from text. The model was first presented in the article Ho et al. [2020] and its architecture is very different from Transformers, since it is based on the use of differential equations to generate from simulated noise. A theoretical substantiation of the model architecture is not provided, since the emphasis is on using transformer models to solve the inverse problem.

To create the dataset, the following steps were taken, the code is given in 4.7.1:

1. Poetry is extracted from the dataset of the formed poetry by keywords.

2. The model for translation from Russian poetry into English is downloaded and applied. This step is necessary because the Stable Diffusion model can only work with English text so far.

   The **facebook/wmt19-ru-en** model described in the article Ng et al. [2019] was used as a translator, where the original transformer with BPE tokenizer integration (which is used in GPT-type models) is used as the main architecture. This version of the model includes only embeddings that allow translation from

Russian into English, excluding other languages stated in the article. According to the estimates given in the article, this model beats previous versions, and still performs well.

3. The Stable Diffusion model is downloaded and sequentially applied to obtain an image based on the translated poetry.

The final version of the dataset can be seen at the link. In the scientific community, the use of generative models is becoming a common practice for collecting datasets, as it allows minimizing money and time resources for obtaining training data. For example, in a recent article Liu et al. [2023], generation using GPT-3.5 is used to create a dataset to check the quality of code generated by models. As a result, the dataset assembled with the help of LLM (large language model) was 81 times larger than the dataset assembled by hand and showed realistic evaluation results.

2.2.2 Training process

One part of the study is an attempt to fundamentally train the general Visual Encoder Decoder model with different parameter for training in size of weights. The **base model** (**Definition 1**) is used as the architecture for training model. All code can be found in 4.8

Training dataset.

To train the model, the simulated image-author-poetry triplets need to be converted into a training dataset format for input to the model.

The general approach for creating a training dataset (**Figure 34**) is as follows:

1. Extracting embeddings from the image to feed them to the model encoder.

   The **openai/clip-vit-base-patch32** processor explained in part 1.7 can be used as a preliminary feature extractor to get input embeddings for further model training. Then the training model will have some meaningful idea of input embeddings and will learn the more difficult task of matching input embeddings with poetry.

   The result is CLIP processor passed with image is pixel values - pre-processed image tensor with corresponding embeddings.

2. Extracting embeddings from text to feed them to the model decoder.

   Using the same tokenizer as for the pre-trained decoder block in **base model** (**ai-forever/rugpt3large-based-on-gpt2**) is relevant.

   Also adding special token $< bos >$ as beginning of sentence, $< sep >$ as a separation token, $< eos >$ as end of sentence token is needed and $< pad >$ as additional token. The output value of the model (and, accordingly, the input to the decoder during training) forms as $< bos >$ author $< sep >$ poetry $< eos >$ and this text should be tokenized.

The tokenizer will return a dictionary containing token indices corresponding to the input sentence and an attention mask to show at what point the semantic sentence ends and empty additional tokens(< *pad* >) begin to equalize the length of incoming embeddings for the model.

3. Also, within the architecture of the encoder-decoder model, it is necessary to designate the output value, the prediction of which will be evaluated during training. To do this, labels are formed like a copy of the received token indices.

4. The return value of the dataset for the current poetry index is a dictionary with image embedding values, embedding token indexes, attention masks, and labels.

Model.

To train the model from the beginning according to the given architecture, it was necessary not to download the pre-trained model, but to load its configuration and randomly initialize the weights as shown in **Figure 35**.

Then the model and dataset are run through the training pipeline (**Figure 36**), within which the dataset is divided into training and validation parts, the data collection for training is initialized, the training parameters are initialized, and the process is started.

FP16 and FP32. As part of the work, two models with different accuracy indicators were trained from scratch:

- **vit-rugpt3-large-poetry-fp16** - model weights are stored in float16, this saves memory during training, but loses training accuracy;

- **vit-rugpt3-large-poetry-fp32** - model weights are stored in float32, which allows to achieve maximum accuracy during training, however, training such a model requires a large amount of memory.

2.2.3 Fune-tuning process

In other solution **base model** was also fine-tuned on the dataset of image-poetry pairs, which requires much less time, but the same amount of resources as when learning from scratch.

The training pipeline remains unchanged, and for fine-tuning process ViT processor was used instead of the CLIP. To download the pre-trained model, the code shown in **Figure 37** was used. This model was named as **vit-rugpt3-large-poetry-ft** for simplisity.

2.2.4 Vit encoder and RuPoetry models

Inspired by the encoder-decoder **base model** architecture, in which two independent pre-trained models are connected to solve the general problem of generating a description from a picture, the idea arose to connect the visual encoder and the advanced decoder similar to **base model**.

As mentioned earlier, in the previous coursework 5 models were fine-tuned to generate poetry in the style of famous poets, but using medium configuration (**ai-forever/rugpt3medium-based-on-gpt2**) unlike large decoder from **base model**. As described in 1.2, these models represent a text decoder from transformers with no cross-attention block.

The big idea is to use a visual encoder (**"google/vit-base-patch16-224-in21k"**) and pair it with the fune-tuned text decoders to get 5 independent visual encoder-decoder models type of **base model**. For such a connection, cross-attention needs to be initialized within the decoder, as explained in part 1.5. These initialized weights need to be optimized (trained) so that the generated poetry matches the input image.

Therefore, there are the following steps for each of the 5 authors (Pushkin, Mayakovsky, Yesenin, Tyutchev, Blok):

1. Just as before, a training dataset is created, where only the poetry for the current author is transmitted to the decoder as input;
2. ViT processor is used as the image processor;
3. The name of the model of the visual encoder and decoder that generates the poetry of the current author is passed to the class of the visual encoder-decoder model, as shown in **Figure 38**;
4. In addition to expanding the embedding space, cross-attention is initialized in the decoder of the model part.
5. Training according to the standard training pipeline.

For simplicity, the designations for this type of models (combinations of pre-trained models for image processing and poetry generation) will be as follows:

- **vit-rugpt3-medium-poet** which mean that this model gen generate poetry from image for a certain poet.

2.3 Evaluation

As part of the evaluation, the trained models for poetry will be evaluated in three areas: assessment of the generation of poems from keywords, assessment of the

correspondence to the style of the generated poetry with real one, and assessment of the correspondence of the input image to the output poetry.

2.3.1 Keywords and poetry

The extraction of poetry from keywords was considered in the context of using a two-stage pipeline to extract poetry from pictures, using a pre-trained model to describe the picture and a pre-finished model to generate poetry based on keywords.

As a result of this decision, an image-keywords-poetry pipeline was formed, where the connection between the input image and the output poetry directly depends on the ability of the model to directly generate verses from keywords.

Model	Cosine Similarity Mean	Cosine Similarity Std
real poetry	0.915	0.025
ruGPT-3 medium	0.910	0.023
ruGPT-3 large	0.910	0.024

Table 7: The similarity of keywords and generated poetry for real poetry (the ability to form keywords is assessed) and for the medium and a large trained model (the ability to generate poems for given keywords is assessed). The score is based on the cosine similarity of keyword embeddings and poetry.

Table 7 shows the average cosine similarity score and its standard deviation between keyword embeddings and embeddings of whole generated or real poetry. For real poetry, it was evaluated how well the KeyBERT library, using Sentence-BERT, is able to extract keywords. Based on the score, on average, the set of extracted keywords is 92% representative of the core essence of all poetry. When training GPT-3 models of different configurations to generate from keywords, the match between the keywords for medium and large model averaged 91 percent, with little difference in standard deviation.

Based on this, it can be concluded that the trained models do a great job of generating poetry from keywords. An example showing the qualities of matching keywords and poetry can be found in **Figure 18**. Note that the keywords can be completely found in different forms in the main sentence and the mood of the poem describes the feeling of the combination of keywords.

2.3.2 Generated and real poetry

The next step is to evaluate how close the generated verses are to real verses in terms of style. To do this, each model for each author generated an amount of poetry corresponding to the size of the original dataset.

Then, with the help of Sentence-BERT trained in Russian, embeddings were generated both for poetry generated by different models and for real poetry. In the **Table**

Автор: Маяковский
Ключевые слова: любовь, жизнь, сон
Поэзия: Жизнь на!
Дай поцелую!
Милая!
Нежная!
В сон!
В жизнь!
Поверь,
лучше нету!
Люблю твой стан упругий,
Твои шелковые космы,
И синие глаза,
И смех сквозь слезы
Твои,
Люблю,
Нежная,
Ириска моя!

Figure 18: An example of generating output poetry based on input keywords in Mayakovsky's style.

8, you can see the average cosine closeness value for the received embeddings for each model and each poet, in order to evaluate how well different models captured the style.

The key findings are as follows:

- A pre-trained model for describing an image in Russian, fine-tuned on the image-author-poetry corpus, **vit-rugpt3-large-poetry-ft** shows the best results for most authors. The model understood the style of Yesenin (88.3% similarity) and Mayakovsky (86.9% similarity) best of all. The similarity with other authors is 81−83%. Interestingly, a pre-trained model of the same decoder configuration (**ruGPT-3-large-key2poetry**) does not achieve this quality of similarity. The examle of poetry can be found at **Figure 19.**
- The closest in similarity to the **vit-rugpt3-large-poetry-ft** model are the **vit-rugpt3-medium-poet** type models, in which two pre-trained models are connected: a visual encoder and a text decoder advanced on poetry od certain poet. Note that for Tyutchev, this model shows the best result in 85.9% similarity.

  It is especially interesting that the decoders of these models have the same architecture, but different configurations (large and medium). This proves the hypothesis that the large model can handle the task of generating different poets at the same time, but the medium model shows the same results by learning from each poet independently.

Model	Mayakovsky	Tyutchev	Blok	Esenin	Pushkin
ruGPT-3-medium-key2poetry	0.778	0.751	0.770	0.755	0.761
ruGPT-3-large-key2poetry	0.781	0.750	0.767	0.757	0.758
vit-rugpt3-medium-poet	0.833	**0.859**	0.803	0.830	0.811
vit-rugpt3-large-poetry-fp16	0.824	0.757	0.761	0.763	0.766
vit-rugpt3-large-poetry-fp32	0.824	0.758	0.762	0.765	0.768
vit-rugpt3-large-poetry-ft	**0.869**	0.821	**0.818**	**0.883**	**0.829**
ruGPT-3 medium poetry	0.789	0.723	0.711	0.713	0.718

Table 8: Cosine similarity calculated from embeddings of real and generated poetry using the presented models for each considered Russian poet.

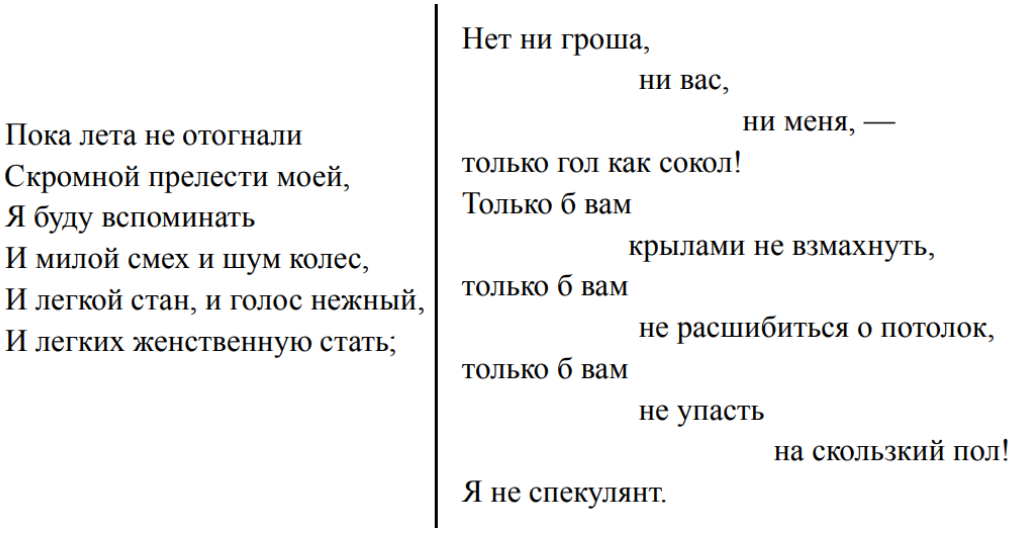


Figure 19: An example of generated poetry in Pushkin's (**left**) and Mayakovsky's (**right**) styles using **vit-rugpt3-large-poetry-ft** model.

- Model-generated poetry using keywords (**ruGPT-3-medium-key2poetry, ruGPT-3-large-key2poetry**) smoothes the results for all poets such that the models are stylistically 75-78 percent similar to real poetry for all poets.

  At the same time, despite the different configuration of the models and the number of parameters and weight, a large model does not show the best result. Perhaps this is due to insufficient training time and a larger model should be trained much longer to achieve a better result.

- Despite the same size of the model (**ruGPT-3-medium-key2poetry**) and the models trained in the last term paper for automatic generation of poetry (**ruGPT-3 medium poetry**) have different similarity indicators by authors.

  The generation model from keywords, despite the simultaneous analysis of all authors, performs better than separate models for each author. This is due to the fact that the first model was trained on a cleaned updated dataset, while the second type of models was trained on a poorly formed dataset.

- In this case, training from scratch almost always shows better results than

models that generate poetry from keywords. It is interesting that the different accuracy estimates in fp16 and fp32 types to store the weights of the models had almost no effect on the final quality of similarity.

However, despite the good scores of similarity with the poetry of real authors, the model produced a completely meaningless text, stylistically looking and reading like a certain author, unlike other models that produced really similar stylistically and meaningful poetry.

И тот не ль?
Друзья мои вечно,
Душу на груди,
В мир внемлет – зреют над главою,
Светить одинокойный
Утеет в тенистой
И свет; о море младость,
Твой беспечной ветеро.
Поют тлеют
На лоне пламенная!
О, что под окном
Поникнут,
По сладострастье свет;
Сердце в дали,
Кудесник свет
Еще ты в синяя,
И ты поносит,
И отчись в нас этот свет.

И вот
урядник
и в буравной и грязи
выгоные
выжигает выпроме.
Ваньше глаза
иззабитой выев
и скат у вас.
Но
не в ответ
с того
попаши
раскерапонпицы
осквозьСтесыжобычь

Figure 20: A bad example of generated poetry in Pushkin's (**left**) and Mayakovsky's (**right**) styles using **vit-rugpt3-large-poetry-fp32** model.

As an example of such generated poetry, one can see the poem of Mayakovsky and Pushkin on the **Figure 20**. Note that the model begins to generate a large number of non-existent words, and if for Pushkin this adds charm, then for Mayakovsky it is almost always an incomprehensible text.

This result is due to the fact that the model learns from scratch to recognize the language and process the image, therefore the model managed to catch the style, but did not learn how natural language works.

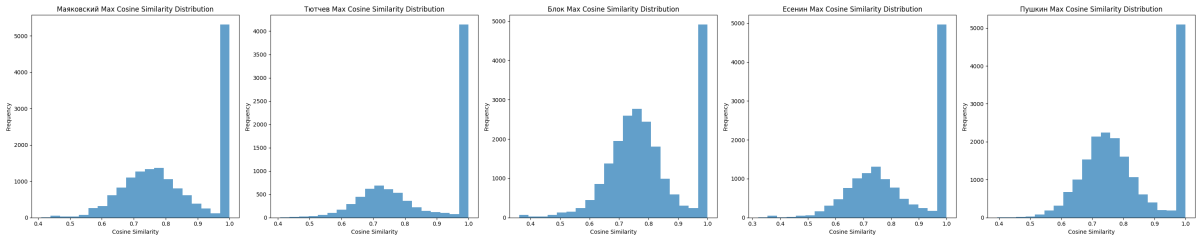


Figure 21: Distribution of cosine similarity for embeddings of real and generated poetry using the **vit-rugpt3-medium-poet** model.

Note that the Cosine Similarity Distribution shown in **Figure 21** over the generated embeddings of real and generated poetry is as follows: most embeddings have a

cosine similarity very close to 1 for each poet, and the rest of the embeddings have completely different similarities. This suggests that the model captures some stylistic features well, while the rest are common (basic) for each author and are not tailored for a specific task.

Model	BLEU-3	DIST-3
Mayakovsky	0.347	0.973
Pushkin	0.312	0.988
Tyutchev	0.3	0.979
Esenin	0.56	0.978
Blok	0.248	0.97

Table 9: Metrics calculated from real and generated poetry using the **vit-rugpt3-large-poetry-ft** model

To assess how well the best model **vit-rugpt3-large-poetry-ft** was trained and detect the diversity rate, the evaluation of the metrics BLEU-3 and DIST-3 is provided, which can be seen in Table 9. The BLEU-3 metric make it clear that the proportion of the presence of 3-grams from real examples in the generated poetry is significant. This indicator is especially high for Esenin, whose model present the greatest cosine similarity, which makes it possible to assume that the model is overfitted on his poetry. However, the assessment of the diversity metric DIST-3, which consists in calculating the proportion of diverse 3-grams in generated poetry relative to the real one, shows that the diversity is very high and close to 1. This proves that the model is not overfitted.

2.3.3 Images and generated poetry

To assess the quality of generating poetry from an image, cosine similarity was also used, calculated from the embeddings of the image and poetry. In this case, in order to better understand which texts are close to the input image and which are far away, the CLIP model was used to obtain embeddings, trained to extract features from both the image and the text.

Model	Image and Poetry similarity
vit-rugpt3-medium-poet	0.217
vit-rugpt3-large-poetry-fp16	0.201
vit-rugpt3-large-poetry-fp32	0.205
vit-rugpt3-large-poetry-ft	**0.263**
real	0.28

Table 10: Cosine similarity calculated from embeddings of image and real or generated poetry using the presented models

The evaluation result presented in Table 11 showed that the pre-trained and fune-tuned **vit-rugpt3-large-poetry-ft** model is the closest to the result of similarity be-

tween real poetry and image, generated by the Stable Diffusion model. This relativity is present in view of the fact that the CLIP model was trained to recognize objects and is able to form classification embeddings well with high accuracy. However, since text is being considered, a value close to 30 indicates a good match.

Esenin	Mayakovsky	Blok
Хорошо лежать в траве зеленой И смотреть, как мимо проплывают облака, Хорошо, когда можно чуточку задремать, Чтобы потом в дреме услыхать Отдаленные раскаты грома.	Утреет. Вставайте, сонные! Ранний петел бодрится, бодрится. На дорогах петухи пропели. Хорошо, что вчера я не поехал в Ялту – на Ай-Петри.	Ты сверкнул в утро дней пустынных, Явился в блеске золотых, Но не к моей блаженной могиле, А к чужой, незнакомой земле. Я умираю, гибну здесь, Там – закат блистательный и новый, А ты всё спешишь, ты всё немеешь, Недавний цвет, вчерашний запах!

Tyutchev	Pushkin
Как порою, как снег, ложится На город тихий зимний сон, — И нет ему исхода, борьбы, Вся жизнь его, как тихий сон.	Метель серебрится, Немолчный звук, То отголосок прежних дней Серебрится в лунном сияньи. Умчались, умчалися прочь Все счастливые дни моей жизни, И сердце остыло, и ум холодный Страшен мне. Брожу один, и в лунном сияньи Мне вспомнилась прежняя жизнь, Метель серебристится, Немолчный звук... Отчего ты, любовь моя, Так грустна и так унылa?

Figure 22: Example of inputs and outputs of **vit-rugpt3-large-poetry-ft** to solve image to poetry task.

Note that the **vit-rugpt3-medium-poet** model shows much lower results. This happened because during the finetuning of two connected pre-trained models, such a combination did not have time to learn matching the image and text due to the small data set. Model **vit-rugpt3-large-poetry-ft** was originally trained for this match and examples of this model can be seen in **Figure 22**.

It can be seen how the mood of poetry changes depending on the image. On the top picture there are poems about nature, colors, the sun, while on the bottom picture there are gray paints, evoking the sadness that poetry reflects. At the same time, the poems capture the key objects in the picture: snow, sky, clouds.

The study tested a large number of approaches. One of the best models for generating lyrics from music is **vit-rugpt3-large-poetry-ft**. It was also necessary to consider the amount of memory required to run a particular solution. The comparison is shown in **Table 11**. Note that the fine-tuned **vit-rugpt3-large-poetry-ft** model is the most memory-optimal for loading locally to access all poets.

Thus, in this chapter, two different approaches to the generation of poetry based

Model	Model size
vit-rugpt3-medium-poet	2GB for each poet
vit-rugpt3-large-poetry-fp16	5 GB
vit-rugpt3-large-poetry-fp32	5 GB
vit-rugpt3-large-poetry-ft	**5 GB**
image-keywords-poetry medium	6,6 GB
image-keywords-poetry large	8,1 GB

Table 11: Model size for each type of models for image to poetry task.

on the image were discussed: two-stage (image-keywords-poetry) and building a full-fledged visual encoder-decoder model. As part of the first approach, generation models from keywords were trained based on the GPT-3 decoder model in different configurations. As part of the second approach, 3 solutions were used: 1. Training from scratch on the architecture of a visual encoder-decoder model that can generate a description for an image, which did not give good results, since the model studied the Russian language from scratch on the corpus of poetry. 2. Fine-tuning such a pre-trained model, which gave the best result, because the model is able to correlate the image with the text, and it was tuned to generate poetry. 3. The use of a pre-trained encoder and a fine-tuned decoder within a common visual encoder-decoder model, which gave a good understanding of the style, but a poor match between image and poetry.

A detailed assessment was also carried out for this 11 models by metrics and a person on the question of the meaningfulness of the generated poetry, and as a result, the best model was chosen that meets not only stylistic requirements, but also knows how to generate text with feeling and sense.

3 Music generation from images

In this part of the work,an approach for training a music generation model based on an incoming image was considered. The main idea is to use a visual encoder-decoder model, the concept of which is described in parts 1.5 with some changes. The whole project with code and link to model can be found at GitHub repository by link https://github.com/anyaschenikova/image2music.

3.1 Data collection

To train the model, it was necessary to collect picture-music pairs in the ABC format. The code for collecting this dataset can be found in 4.7.2.

For copyright reasons, the dataset declared in the article on generating music from text Wu and Sun [2023], consisting of text-music pairs and described in part 1.6.2, has not been publicly published. Instead, the **WikiMusicText (WikiMT)** dataset was provided, which includes 1010 pairs of text and music data and can be used to evaluate the performance of language and music models.

This dataset has several columns: title, artist, genre, music in paragraph notation and a short description. The description is a historical reference about the creator, performer and role in the musical world and musical features.

However, if the model is trained to generate music from such a text, then the model will qualitatively generate music with strict restrictions on genre and direction. What experiments with a fine-tuned BART model prove is thatm generalized abstractions or just poetry, do not allow to produce a qualitative output value.

Accordingly, it was not possible to use Stable Diffusion to generate pictures from the description, since the description do not provide any idea about the images inspired by this music.

3.1.1 GPT-3.5 role

Based on the information about the image, it was decided to send a request to GPT-3.5 in order to collect this kind of dataset.

The GPT-3.5 model is a large language model (LLM) from OpenAI, whose weights are not publicly available. This model is similar to GPT-2 in architecture, however, for its training, a learning-with-consumption or Reinforcement-learning approach with the integration of human evaluation was used to train model to chat or communicate with people in a dialog form.

In order to generate imaginative stories about the current music in question for this language model, it was necessary to give the model the appropriate instructions presented in **Figure 23**. At the end of the input request, a small batch was sent

```
Help me to fune-tune model. In list below you can see strings with info about song.
Create story about feelings, inspired by each song.
Requirements:
1.Return story for each line in folowing format:

SONG NAME:...
STORY:...

2. You must to divide each song name and story by ###.
3. Avoid mention of sertain song in story, provide only general info in story.

List of songs info:
{lines_list}
```

Figure 23: Schema propt to the GPT-3.5 large language model for creating a story about music-inspired images based on historical information.

with information about different melodies the and model returned a story for each composition.

As a result of this stage, for each music in the ABC notation, a sensory story was generated about the images and emotions inspired by the music. The code for creating queries is shown in **Figure 33**.

3.1.2 Stable Diffussion

Based on the formed music story, an image is formed with the help of Stable Diffussion and code for this can be found in Appendix 4.7.1.

It should be noted that the stories formed with the help of GPT-3.5 were sometimes reduced to describing the music itself (genre, style), so most of the collected pictures in the dataset appear as singing people in different interpretations. However, this did not prevent from getting an interesting result in terms of generating music from an image.

The final version of collected dataset with image, genre and music in ABC notation columns can be found by link.

3.2 Visual encoder-decoder model

The main idea of the model is to use a visual encoder and connect it to a pre-trained generative model that is a text decoder like it was explained 2.2.4. The unusual solution in this case is that the visual transformer **"google/vit-base-patch16-224-in21k"** is connected to the BART-based **sander-wood/text-to-music** model, which in turn is already an encoder-decoder model, advanced for the task of generating music from text.

Instead of the initial encoder layers in the BART model the layers and weights of

the visual encoder ViT model will be used, which can be seen in architecture of initialized model. The code for initializing such a visual encoder-decoder model can be found in **Figure 39**. Cross-attention in this case will also be randomly initialized, which means that it is necessary to optimize these model weights using additional training on the image-music corpus.

The learning steps remain the same for the entire process described above in the part of poetry, however, with the only modification: only music in the ABC format is fed to the text tokenizer from the BART model. The ViT image processor was also used as a visual extractor.

3.3 Evaluation

In this part, the trained model will be evaluated for the task of generating music based on the input image. The same approach was used for evaluation as for poetry. Using the CLIP model, music embeddings and corresponding image embeddings were calculated.

Model	Image and Music similarity
vit-bart-image2music	0.245
real	0.252

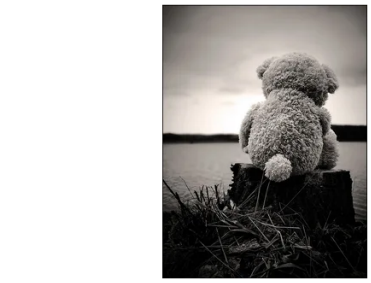
Table 12: Cosine similarity calculated from embeddings of image and real or generated music using the vit-bart models

The results are presented in **Table 12**. It can be seen that the model has learned to generate music from the image in the same way as real poetry and music are compared. A cosine similarity close to 30 indicates a good match quality in terms of embeddings.

Genre	Count of examples
Pop	225
Jazz	212
Country	167
R and B	109
Rock	98
Folk	95
Dance	75
Latin	22

Table 13: Distribution of genres in the training dataset for the task of generating from pictures to music in ABC notation.

In evaluation of the sound of music, it became noticeable experimentally that the dataset contains the most positive versions for musical genres (**Table 13**), Pop and



```
X:1
L:1/4
M:4/4] C |:"F" F2"C7" G2 |"F" A2 z A/A/ |"G7" A G F E | D2 z C/C/ | D E F G |
A3/2 ^G/ A c |
"C" c4- | c3 c |"Bb" d d d e | f2 z F/E/ | F G A B | c2 z c/c/ | c c"Gm7" c d |"C e2 z
e/e/ |
 e e e d | c4 |"Dm" d4 | d2 z2 | z d c B | A A A ^G | A4 | z4 | F2 G A | B A B c | d c d
c | B4 |
 z2 z D | D D D E | F F/F/- F G/F/ | C C C D | E F E F | E D C C/B,/ | A, A,/A,/ A, B,
| C3 A, | F,4 |:
```

```
X:1
L:1/4
M:4/4min
K:F
V:1 treble  (1  2  )" z2 A2 |"F" c3 A | F2 G A |"Gm" B2 G2- | G2 B2 | d3 d | d2 e d
|"C7" d c3- |
 c2 z c |"Bb" f2 d c | d c B _A |"Eb7" G2 F E | G F E _D |"Dm" D4- | D4 | z2"F7" z
A/A/ ||
"Bbmaj7" A2 _A G | _G2 F =E |"Ab7" F3 E | _E3 E/E/ |"Db7" _D2 _D C | D F _A
_d |"Gb9" c2 _c B | _A4 |
"C9" G3 ^G | A2 A G | ^G2 =G ^F | G4 | c3 c | B2 _B A |"Fmaj7""^S   " - - - -" F B
"F"
```

Figure 24: Image to music example in ABC notation.

Jazz are most represented. Therefore, the model is skewed more to generate positive melodies with an interesting weave of melodies.

Examples of image for generation can be found on **Figure 24** and the music it generated can be listened by links: above image and below in online editors. Analyzing the quality of the generated music, the model generates a rhythmic, melodic and naturally interesting musical text, but due to the prevalence of positive style in the dataset, it does not cope well, but acceptable with the task of generating music in accordance with the image that causes sad emotions.

Speaking about the quality of the generated poetry: the model does not always give a good result, or it is retrained and then there is no diversity. This problem can be corrected by a additional selection of data to correlate with non-positive images. Note that the playback speed is not adjusted in any way within the generated text, ince the AB format is an alternative to musical notation, in which the speed of performance is given by certain terms and the terms can be perceived subjectively, not definitely exactly. So the output text may be logical in terms of perception, but changing the playback speed will give its results. This is especially clearly seen on the famous composition from the cartoon ”The Lion King”, the voice acting of which is in the recorded ABC notation and can be found at the link.

In conclusion, in this chapter, an approach was considered for training a visual encoder-decoder model, which consists in combining two pre-trained models: a visual transformer encoder in the form of a ViT model and fune-tuned BART encoder-decoder model capable of generating music based on a limited input text. While obtaining a common visual encoder decoder model, the encoder block of the BART model was replaced by the ViT model. The general model was trained on the generated image-music text corpus in ABC notation, which made it possible to solve the problem of music generation as a problem of generating a specific text. As a result,

the correspondence of the generated music and the input image is very close to the correspondence of the real music and the generated Stable Diffusion image according to the colorful description. However, due to the unevenly collected dataset, music generation did not always give a stylistically accurate result, which gives ground for further reflection on the possibilities of developing this idea.

Conclusion

In this thesis, the following things were done:

1. Key technologies for solving the problem of generating poetry and music based on images were studied and explained in detail in the theoretical part of the work, namely:
    - The Encoder-Decoder model and device of the Transformers was revised, which laid the foundation for the current work;
    - A detailed description was given to the architecture of the visual encoder ViT model, capable of extracting features from an image with high quality;
    - Technologies for extracting keywords from text, with detailed consideration of BERT and Sentence-BERT technologies, were presented and evaluated;
    - It was reminded about the device decoder of the GPT-3 model, which was upgraded for the generation of verses in the previous coursework;
    - An encoder-decoder BART model for generating music from text was considered, which laid the foundation for a solution for generating music from an image;
    - A CLIP processor was also considered, trained to detect proximity between an image and text, used as a process and as a model for assessing the correspondence between an input image and output poetry or music.
2. As part of the practical part, based on the described theoretical explanation, several architectural solutions were proposed for the problem of generating poetry from an image, namely:
    - In a multi-stage approach, consisting of a model that converts an image into keywords, and then a model that converts them into poetry for different configurations, showed a match between keywords and poetry in 91% of the cosine proximity of the corresponding embeddings;
    - In learning from scratch the combined architecture of a visual encoder and a text decoder, despite the high rate of stylistic proximity (81-83%) of the generated poetry to the real one, the model did not have time to learn the language from scratch and generated meaningless text;
    - Models that combine a pre-trained visual encoder and a fine-tuned text decoder on poetry produce excellent stylistic affinity (80-86%), but poor image match;
    - The image to text model, fune-tuned on the image-poetry corpus, showed itself best of all, since the results in terms of semantic similarity (82-88%) and correspondence of the input image to the output poetry were the highest.

3. To solve the problem of generating from image to music, the approach of connecting two pre-trained models was used, which showed an acceptable result of matching and generation. However, the prevalence of positive melodies in the dataset does not have the best effect on the possibility of high-quality music generation based on non-positive images, which leads either to a mismatch with the image, or to a violation of the rhythmic and melodic requirements for the output music.

4. Also, as part of the practical part, the picture-poetry, picture-music and poetry-keywords datasets were collected to train visual encoder-decoder architecture models, however, the dataset with music needs further improvement.

In conclusion, a lot of work has been done to obtain high-quality results in solving the problem of generating poetry and music from an image. With the development of technology, there are more and more opportunities to study this problem, and only recently a large number of technologies have appeared that can be also tried to use for this complicated task.

References

Jean-Pierre Briot, Gaëtan Hadjeres, and François-David Pachet. Deep learning techniques for music generation – a survey, 2019. pages 5

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020. pages 5

Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers, 2020. pages 14

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019. pages 5, 21

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale, 2020. pages 15, 16, 19

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models, 2020. pages 39

Ting-Hao Kenneth Huang, Francis Ferraro, Nasrin Mostafazadeh, Ishan Misra, Aishwarya Agrawal, Jacob Devlin, Ross Girshick, Xiaodong He, Pushmeet Kohli, Dhruv Batra, C. Lawrence Zitnick, Devi Parikh, Lucy Vanderwende, Michel Galley, and Margaret Mitchell. Visual storytelling. In **Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies**, pages 1233–1239, San Diego, California, June 2016. Association for Computational Linguistics. doi: 10.18653/v1/N16-1147. URL `https://aclanthology.org/N16-1147`. pages 5

Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Joan Puigcerver, Jessica Yung, Sylvain Gelly, and Neil Houlsby. Big transfer (bit): General visual representation learning, 2019. pages 19

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension, 2019. pages 6, 29, 30

Jiwei Li, Michel Galley, Chris Brockett, Jianfeng Gao, and Bill Dolan. A diversity-promoting objective function for neural conversation models. In **Proceedings of**

the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pages 110–119, San Diego, California, June 2016. Association for Computational Linguistics. doi: 10.18653/v1/N16-1014. URL https://aclanthology.org/N16-1014. pages 28

Juntao Li, Yan Song, Haisong Zhang, Dongmin Chen, Shuming Shi, Dongyan Zhao, and Rui Yan. Generating classical Chinese poems via conditional variational autoencoder and adversarial training. In **Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing**, pages 3890–3900, Brussels, Belgium, October-November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1423. URL https://aclanthology.org/D18-1423. pages 5

Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft coco: Common objects in context, 2015. pages 38

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation, 2023. pages 40

Nathan Ng, Kyra Yee, Alexei Baevski, Myle Ott, Michael Auli, and Sergey Edunov. Facebook fair's wmt19 news translation task submission, 2019. pages 39

Kishore Papineni, Salim Roukos, Todd Ward, and Wei Jing Zhu. Bleu: a method for automatic evaluation of machine translation. 10 2002. doi: 10.3115/1073083.1073135. pages 5, 28

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision, 2021. pages 30, 31

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer, 2019. pages 34

Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks, 2019. pages 23

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In **Proceedings of the 31st International Conference on Neural Information Processing Systems**, NIPS'17, page 6000–6010, Red Hook, NY, USA, 2017. Curran Associates Inc. ISBN 9781510860964. pages 5, 7, 10, 14

Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. Show and tell: A neural image caption generator, 2015. pages 5

Xiaolong Wang, Ross Girshick, Abhinav Gupta, and Kaiming He. Non-local neural networks. In **Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)**, June 2018. pages 14

Shangda Wu and Maosong Sun. Exploring the efficacy of pre-trained checkpoints in text-to-music generation task, 2023. pages 27, 28, 50

Qizhe Xie, Minh-Thang Luong, Eduard Hovy, and Quoc V. Le. Self-training with noisy student improves imagenet classification, 2019. pages 19

4 Appendix

4.1 Activation functions

4.1.1 ReLU activation

The ReLU activation function is differentiable at all points except zero. For values greater than zero, for simplicity the maximum of the function. This can be written as:

$$ReLU(X) = max(0, X)$$

4.1.2 GELU activation

The Gaussian Error Linear Unit, or GELU, is an activation function. The GELU activation function is $x\Phi(x)$, where $\Phi(x)$ the standard Gaussian cumulative distribution function. The GELU nonlinearity weights inputs by their percentile, rather than gates inputs by their sign as in ReLUs ($x\mathbf{1}_{x>0}$). Consequently the GELU can be thought of as a smoother ReLU.

$$\text{GELU}(x) = xP(X \leq x) = x\Phi(x) = x \cdot \frac{1}{2}\left[1 + \text{erf}(x/\sqrt{2})\right],$$

if $X \sim \mathcal{N}(0, 1)$.

One can approximate the GELU, but PyTorch's exact implementation is sufficiently fast such that these approximations may be unnecessary.

4.2 Positional embeddings for ViT

"5 shots linear" means that the model was scored based on how well a linear classifier trained on features generated by the model with only 5 examples per class was able to perform on the test set. This is a strict quality check of the features learned by the model, since the classifier is not allowed to update the features and must be content with just a few examples.

4.3 Cosine Similarity

Cosine similarity measures the similarity between two vectors of the inner product space. It is measured by the cosine of the angle between two vectors and determines whether the two vectors are pointing in roughly the same direction. It is often used to measure the similarity of documents in text analysis. The formula is:

$$\text{cosine similatiry} = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \cdot \|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \sqrt{\sum_{i=1}^{n} B_i^2}},$$

where **A** and **B** are vectors with n components.

4.4 Softmax function

The soft max function is used to normalize a vector into a probability distribution. The returned values are summed up to one and can be used, for example, to predict the next token in the offer with the highest probability.

Given the vector $\mathbf{X} = [x_1, x_2, ..., x_n]$, the softmax function formula can be written as:

$$\text{Softmax}(x_i) = \frac{exp(x_i)}{\sum_j = 1^n (exp(x_j))}, i = 1, \ldots, n$$

The output values of the function lie between 0 and 1.

4.5 Architecture for Sentence-BERT

```
SentenceTransformer(
        (0): Transformer({
      'max_seq_length': 128,
      'do_lower_case': False}) with Transformer model: XLMRobertaModel
        (1): Pooling({
      'word_embedding_dimension': 1024,
      'pooling_mode_cls_token': False,
      'pooling_mode_mean_tokens': True,
      'pooling_mode_max_tokens': False,
      'pooling_mode_mean_sqrt_len_tokens': False})
)
```

Figure 25: Architecture of Sentence-BERT for keywords extraction.

4.6 Poetry from keywords

4.6.1 Keywords extraction

```python
from keybert import KeyBERT
from sentence_transformers import SentenceTransformer

model = SentenceTransformer("xlm-r-large-en-ko-nli-ststb")
keyword_extractor = KeyBERT(model=model)

def extract_keywords_transformer(text, num_keywords=5):
    text = normalize_and_remove_stop_words(text)
    keywords = keyword_extractor.extract_keywords(
        text,
        keyphrase_ngram_range=(1, 1),
        stop_words=None,
        top_n=num_keywords,
        use_maxsum=True,
        use_mmr=True,
        diversity=0.7
    )
    return [keyword[0] for keyword in keywords]
```

Figure 26: Code for generating keywords from text using KeyBERT and Sentence-BERT

4.6.2 Fine-tuning GPT-3

```
from transformers import TrainingArguments, Trainer, TextDataset, \
DataCollatorForLanguageModeling, GPT2LMHeadModel, AutoTokenizer
import torch
import pandas as pd


# download model and tokenizer
tokenizer = AutoTokenizer.from_pretrained(path_to_medium_model)
model = GPT2LMHeadModel.from_pretrained(path_to_medium_model).to(DEVICE)

#adding special tokens and resize model embeddings size
SPECIAL_TOKENS = {'bos_token' : "<bos>", "eos_token" :"<eos>", 'pad_token':'<pad>'}
tokenizer.add_special_tokens(SPECIAL_TOKENS)
model.resize_token_embeddings(len(tokenizer))
model.config.bos_token_id = tokenizer.bos_token_id

#creating dataset for training
class myDataset(Dataset):

  def __init__(self, data: pd.DataFrame, tokenizer, gpt2_type="gpt2", max_length=150):

    self.tokenizer = tokenizer
    self.input_ids = []
    self.attn_masks = []

    for ind in data.index:

      author = '<bos> Автор: ' + data.iloc[ind]['autor'] + '\n'
      keywords = 'Ключевые слова: ' + ', '.join(data.iloc[ind]['keywords'].split("'")[1:-1:2])+'\n'
      poetry = 'Поэзия: ' + data.iloc[ind]['text'] + '<eos>'

      form = author + keywords + poetry
      encodings_dict = tokenizer(form,
                                 truncation=True,
                                 max_length=max_length,
                                 padding="max_length")

      self.input_ids.append(torch.tensor(encodings_dict['input_ids']))
      self.attn_masks.append(torch.tensor(encodings_dict['attention_mask']))

  def __len__(self):
    return len(self.input_ids)

  def __getitem__(self, idx):
    return {
        'input_ids': self.input_ids[idx],
        'attention_mask': self.attn_masks[idx]
    }

train_dataset = myDataset(dataset, tokenizer)

#adding data collator for process data during training
data_collator = DataCollatorForLanguageModeling(tokenizer=tokenizer, mlm=False)

#adding trainin arguments
training_args = TrainingArguments(
    output_dir=f'./checkouts/', #The output directory
    overwrite_output_dir=True, #overwrite the content of the output directory
    num_train_epochs=65, # number of training epochs
    per_device_train_batch_size=6, # batch size for training
    warmup_steps=150,# number of warmup steps for learning rate scheduler
    gradient_accumulation_steps=5, # to make "virtual" batch size larger
    save_steps = 5000,
    fp16=True #half-precision training
    )

#training
trainer = Trainer(
    model=model,
    args=training_args,
    data_collator=data_collator,
    train_dataset=train_dataset,
    optimizers = (torch.optim.AdamW(model.parameters(),lr=1e-5),None)
)
trainer.train()
```

Figure 27: Code for fine-tuning ruGPT-3 model for generating poetry from all poets and with adding keywords

4.6.3 Generation pipeline code

```
path_image2text = 'tuman/vit-rugpt2-image-captioning'

model = VisionEncoderDecoderModel.from_pretrained(path_image2text)
feature_extractor = ViTFeatureExtractor.from_pretrained(path_image2text)
tokenizer = AutoTokenizer.from_pretrained(path_image2text)
model.to(device)

max_length = 20
num_beams = 4
temperature = 1.2
gen_kwargs = {"max_length": max_length, "num_beams": num_beams, 'temperature': temperature}

def predict_caption(url):

  image = Image.open(requests.get(url, stream=True).raw)

  pixel_values = feature_extractor(images=image, return_tensors="pt").pixel_values
  pixel_values = pixel_values.to(device)

  output_ids = model.generate(pixel_values, **gen_kwargs)

  preds = tokenizer.batch_decode(output_ids, skip_special_tokens=True)
  preds = [pred.strip() for pred in preds][0]
  return preds
```

Figure 28: Code for initialization **tuman/vit-rugpt2-image-captioning** for generating poetry from image in solution image-keywords-poetry

```python
# Initialize the MorphAnalyzer and load the Russian stop words
morph = MorphAnalyzer(lang='ru')
stop_words = set(stopwords.words("russian"))

def normalize_and_remove_stop_words(text):
    # Tokenize the text
    words = word_tokenize(text, language="russian")

    # Lemmatize and remove stop words
    normalized_words = []
    for word in words:
        parsed_word = morph.parse(word)[0]
        lemma = parsed_word.normal_form
        if lemma not in stop_words:
            normalized_words.append(lemma)

    # Combine the normalized words into a single string
    normalized_text = ' '.join(normalized_words)

    normalized_text =

    return normalized_text

# Example usage
text = "Был солнечный день, и птицы пели весело на ветках."
normalized_text = normalize_and_remove_stop_words(text)
print(normalized_text)
```

Figure 29: Functions for deleting stop-words to form keywords from the output of vision model

```
from transformers import AutoTokenizer, AutoModelForCausalLM

path = 'AnyaSchen/rugpt3-mayak-keywords'
tokenizer = AutoTokenizer.from_pretrained(path)
model = AutoModelForCausalLM.from_pretrained(path)

def generate_poetry(input: str, model, num_beams=3):
  input = input if len(input) > 0 else tokenizer.bos_token
  input_ids = tokenizer.encode(input, return_tensors="pt").to(device)
  # Create an attention mask
  attention_mask = (input_ids != tokenizer.pad_token_id).float()

    # Set the pad_token_id
  tokenizer.pad_token_id = tokenizer.eos_token_id
  with torch.no_grad():
        out = model.generate(input_ids,
                             do_sample=True,
                             num_beams=num_beams,
                             temperature=2.0,
                             top_p=0.9,
                             max_length = 200,
                             eos_token_id=tokenizer.eos_token_id,
                             bos_token_id=tokenizer.bos_token_id,
                             attention_mask=attention_mask
                             ).to(device)
  return tokenizer.batch_decode(out, skip_special_tokens=True)[0]


inp = f'''Автор: Тютчев
Ключевые слова: {', '.join(keywords)}'''
print(generate_poetry(inp, model))
```

Figure 30: Code for generating poetry from all poets and with adding keywords

4.7 Datasets code

4.7.1 Image2poetry

```
from transformers import FSMTForConditionalGeneration, FSMTTokenizer
mname = "facebook/wmt19-ru-en"
tokenizer = FSMTTokenizer.from_pretrained(mname)
model = FSMTForConditionalGeneration.from_pretrained(mname).to(device)

def translate_poetry(input):
  input_ids = tokenizer.encode(input, return_tensors="pt").to(device)
  outputs = model.generate(input_ids)
  decoded = tokenizer.decode(outputs[0], skip_special_tokens=True)
  return decoded
```

Figure 31: Code for downloading and using the model to generate a translation from Russian into English to collect a dataset for the image to poetry task.

```
from diffusers import StableDiffusionPipeline, EulerDiscreteScheduler
import torch

model_id = "stabilityai/stable-diffusion-2-1-base"

scheduler = EulerDiscreteScheduler.from_pretrained(model_id, subfolder="scheduler")
pipe = StableDiffusionPipeline.from_pretrained(model_id, scheduler=scheduler, torch_dtype=torch.float16)
pipe = pipe.to(device)

prompt = 'Vlas Progulkin is a lovely boy, went to bed with a journal. Everything in the journal is interesting. - I
read everything, even a crack! - Neither his father nor his mother could sleep.'
image = pipe(prompt).images[0]
```

Figure 32: Code for downloading and using the model to generate images Stable Diffusion to collect a dataset for the image to poetry task.

4.7.2 Image2music

```
api_key = os.getenv("OPENAI_API_KEY", api_key)
openai.api_key = api_key

def ask_gpt35(prompt):
    response = openai.ChatCompletion.create(
        model="gpt-3.5-turbo",
        messages=[{"role": "user", "content": prompt }],
        max_tokens = 2000,
        n=5,
        stop=None,
        temperature=0.8,
    )
    message = response.choices[0]['message']['content']
    return message
```

Figure 33: Code for requesting the API-GPT-3.5 model to generate a dataset for solving the problem of generating music from an image.

4.8 Training pipeline

```python
from torch.utils.data import Dataset
from transformers import CLIPProcessor, AutoTokenizer, ViTImageProcessor

class ImagePoetryDataset(Dataset):
    def __init__(self, dataset, processor, tokenizer):
        self.dataset = dataset
        self.processor = processor
        self.tokenizer = tokenizer

    def __len__(self):
        return len(self.dataset)

    def __getitem__(self, idx):
        # Load and preprocess the image
        image = self.dataset[idx]['image'].convert("RGB")
        inputs = self.processor(images=image, return_tensors="pt", padding=True)
        pixel_values = inputs["pixel_values"].squeeze(0)

        # Concatenate author and poetry with separator
        text = f"<bos> {self.dataset[idx]['author']} <sep> {self.dataset[idx]['poetry']} <eos>"

        # Tokenize the combined text
        tokens = self.tokenizer(text, return_tensors="pt", padding="max_length",\
                                max_length=128, truncation=True)
        input_ids = tokens["input_ids"].squeeze(0)
        attention_mask = tokens["attention_mask"].squeeze(0)

        # Copy the input IDs to use as labels
        labels = input_ids.clone()

        return {
            "pixel_values": pixel_values,
            "input_ids": input_ids,
            "attention_mask": attention_mask,
            'labels': labels
        }

# Load the CLIP processor
processor = CLIPProcessor.from_pretrained("openai/clip-vit-base-patch32")
# Or use ViTImageProcessor
processor = ViTImageProcessor.from_pretrained("google/vit-base-patch16-224-in21k")

# Load a GPT tokenizer for the Russian language
tokenizer = AutoTokenizer.from_pretrained('ai-forever/rugpt3medium_based_on_gpt2')

SPECIAL_TOKENS = {
  'bos_token': "<bos>",
  "eos_token": "<eos>",
  'pad_token': '<pad>',
  'sep_token': '<sep>'
}
tokenizer.add_special_tokens(SPECIAL_TOKENS)
# Create the Dataset
dataset = ImagePoetryDataset(data, processor, tokenizer)
```

Figure 34: Code for creating dataset object for training image2poetry model, where image feature extractor used CLIP processor (or ViT processor) for creating image token embeddings and tokenizer from GPT-3 is applied to poetry

```
# Create a new VisionEncoderDecoder model with the config
model = VisionEncoderDecoderConfig.from_pretrained("tuman/vit-rugpt2-image-captioning")
model = VisionEncoderDecoderModel(config = config).to(device)

model.decoder.resize_token_embeddings(len(tokenizer))
```

Figure 35: Initialization of Encoder-Decoder ViT-GPT3 model with random weights for training and getting **vit-rugpt3-large-poetry-fp16** or **vit-rugpt3-large-poetry-fp32** models.

```
# Split the dataset into train and validation sets
train_size = int(0.9 * len(dataset))
val_size = len(dataset) - train_size
train_dataset, val_dataset = random_split(dataset, [train_size, val_size])

# Create data loaders
batch_size = 3
train_dataloader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True, num_workers=4)
val_dataloader = DataLoader(val_dataset, batch_size=batch_size, shuffle=False, num_workers=4)

# Define the training arguments
training_args = TrainingArguments(
    output_dir="./checkouts",
    num_train_epochs=27,
    per_device_train_batch_size=batch_size,
    per_device_eval_batch_size=batch_size,
    gradient_accumulation_steps=10,
    evaluation_strategy="epoch",
    logging_dir="./image_poetry_logs",
    save_steps = 1000,
    learning_rate=3e-5,
    weight_decay=0.01,
    # fp16=True,  # Use mixed precision training

# Create a Trainer instance
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=val_dataset,
)

# Train the model
trainer.train()
```

Figure 36: General pipeline for training/fine-tuning process with: splitting data into train and validation sets, loading DataCollators, initialisation of training parameters and Trainer object.

```
# Create a new VisionEncoderDecoder model with the pre-trained model
model = VisionEncoderDecoderModel.from_pretrained("tuman/vit-rugpt2-image-captioning")
model.to(device)
model.decoder.resize_token_embeddings(len(tokenizer))
```

Figure 37: Initialization of Encoder-Decoder model with pre-trained weights for fine-tuning and getting **vit-rugpt3-large-poetry-ft** model.

```
decoder_path = f'AnyaSchen/rugpt3_{info_author[author]}'
model = VisionEncoderDecoderModel.from_encoder_decoder_pretrained(
      "google/vit-base-patch16-224-in21k", decoder_path
  )


model.decoder.resize_token_embeddings(len(tokenizer))
model.config.decoder_start_token_id = tokenizer.bos_token_id
model.config.pad_token_id = tokenizer.pad_token_id
```

Figure 38: Initialization of Encoder-Decoder **vit-rugpt3-medium-poet** model with using pre-trained ViT model as encoder and fine-tuned GPT3-medium model, which generate poetry in style of certain poet.

```
# Define the encoder and decoder names
encoder_name = 'google/vit-base-patch16-224-in21k'
decoder_name = 'sander-wood/text-to-music'

# Create a VisionEncoderDecoderModel from two pre-trained
model = VisionEncoderDecoderModel.from_encoder_decoder_pretrained(
    encoder_name, decoder_name).to(device)

#Additional settings of model config
model.decoder.resize_token_embeddings(len(tokenizer))
model.config.decoder_start_token_id = tokenizer.bos_token_id
model.config.pad_token_id = tokenizer.pad_token_id
```

Figure 39: Initialization of Encoder-Decoder **vit-bart-image2music** model with using pre-trained ViT model as encoder and fine-tuned BART model, which generate music from text.