TÉCNICO LISBOA

# Automatic generation of learning resources: A case study on the generation of true/false sentences.

**Eduardo Moreira Miranda**

Thesis to obtain the Master of Science Degree in

## Computer Science and Engineering

Supervisors: Prof. Claudia Martins Antunes
Prof. Andreas Miroslaus Wichert

### Examination Committee

Chairperson:
Supervisor: Prof. Claudia Martins Antunes
Member of the Committee: Prof. Anna Carolina Nametala Finamore do Couto

**October 2024**

**Declaration**
I declare that this document is an original work of my own authorship and that it fulfills all the requirements of the Code of Conduct and Good Practices of the Universidade de Lisboa.

# Acknowledgments

I would like to thank my supervisors, Prof. Cláudia Antunes and Prof. Andreas Wichert. Their guidance has played a crucial role in shaping this thesis. I would also like to thanks my family and friends for their support, friendship, and care throughout the years. Their presence has been invaluable, and without them, this thesis would not have been possible.

# Abstract

In recent years, the education landscape has undergone a transformative shift towards massification, characterized by the widespread availability of educational resources facilitated by technological advancements. The surge of Massive Open Online Courses (MOOCs) has underscored the need for a continuous supply of learning resources, leading to a burgeoning research focus on automatic question generation. Our primary objective is to address this need by automating the generation of questions related to data charts, with a specific emphasis on crafting true or false data science questions. We explore various models designed for visual question generation, categorizing them based on specific dimensions. In response to this exploration, we propose the development of a dataset comprising triples (Image-question-answer) and the introduction of two models: one dedicated to question generation and another for answering those questions. Model evaluation were conducted using machine translation metrics for the question generation aspect and standard evaluation metrics for binary classification tasks in the context of question answering.

# Keywords

Visual Question Generation; Visual Question Answering; MOOCs; Transformers; LLM; Learning Resources

# Resumo

Nos últimos anos, o panorama da educação tem passado por uma transformação significativa em direção à massificação, caracterizada pela ampla disponibilidade de recursos educacionais facilitada pelos avanços tecnológicos. O aumento dos Cursos Online Abertos e Massivos (MOOCs) destacou a necessidade de um fornecimento contínuo de recursos de aprendizagem, levando a uma crescente investigação sobre a geração automática de perguntas. O nosso principal objetivo é responder a essa necessidade automatizando a geração de perguntas relacionadas com gráficos de dados, com ênfase específica na elaboração de perguntas de verdadeiro ou falso de ciência de dados. Exploramos vários modelos concebidos para a geração de perguntas visuais, categorizando-os com base em dimensões específicas. Em resposta a essa exploração, propomos o desenvolvimento de um conjunto de dados composto por triplos (Imagem-pergunta-resposta) e a introdução de dois modelos: um dedicado à geração de perguntas e outro para respondê-las. A avaliação dos modelos será realizada utilizando métricas de tradução automática para a geração de perguntas e métricas de avaliação padrão para tarefas de classificação binária no contexto da resposta de verdaderio e falso a perguntas.

# Palavras Chave

Geração de Perguntas Visuais; Resposta a Perguntas Visuais; MOOCs; Transformers; LLM; Recursos de Aprendizagem

# Contents

# List of Figures

# Listings

# 1

# Introduction

In recent years, the field of education has witnessed a transformative shift towards massification, marked by the widespread accessibility of educational resources driven by technological advancements. This trend, facilitated by online learning platforms and open educational resources, has the potential to revolutionize traditional educational practices by promoting inclusivity and flexibility. However, it introduces challenges such as larger class sizes, diminishing individual interaction, and overwhelming students with choices [20], requiring more personalized learning pathways. Personalization becomes crucial to address disparities, allowing learners to progress at their own pace [21] and in alignment with their unique learning styles, thereby fostering a more equitable and inclusive educational landscape.

One notable manifestation of education massification is the widespread adoption of Massive Open Online Courses (MOOCs) [22], serving as a means to disseminate course materials online. MOOC platforms have gained prominence, especially with the rise of blended learning, combining face-to-face instruction with digital education [23]. Despite the popularity of MOOCs, challenges persist, and personalization has become a focal point of research [24]. Personalized questions, in particular, offer significant advantages as they can be tailored to individual needs, addressing specific knowledge gaps. Consequently, as observed by Kurdi et al. [25].

There is a growing demand for automated tools that generate learning resources, including exercises, tailored to each user's unique requirements. MOOCs that delve into data-driven subjects, as mentioned

earlier, could greatly benefit from tools capable of automatically generating questions about data charts.

Our main objective in this work is to address this problem by proposing along with a methodology to automatically generate true or false questions based on data charts and a methodology to train a model able to answer those questions.

In this document, we describe the creation of the dataset used to train our models. We propose a process that leverages pretrained image-to-text models capable of visual understanding and optical character recognition, such as Pix2Struct and GiT, by fine-tuning them on diverse data charts (e.g., box-plots, histograms, and decision trees). This approach allows us to extract and transform the necessary information into text format to generate questions related to these data charts. The extracted information is formatted into a prompt, which is then fed into a generative pretrained model, such as GPT-3.5, which includes training in the data science domain. This model subsequently generates the questions. The question-answering process follows a similar approach by leveraging pretrained image-to-text models that extract and transform information into text format, which is then fed to a generative pretrained model with domain knowledge to answer the questions as true or false.

We achieved excellent results in extracting visual information, with scores of 0.9 and 0.85 on the BLEU and ROUGE metrics, respectively. For the question generation task, we obtained scores of 0.85 and 0.89 on the BLEU and ROUGE metrics, respectively. Additionally, we received an average score of 4.4 out of 5 in a manual evaluation conducted by a domain specialist. For the question answering task, we achieved scores of 0.89 and 0.95 on the BLEU and ROUGE metrics for the visual task and an accuracy of 0.85 in answering the questions.

In pursuit of this goal, we explore related work in the field of Visual Question Generation, a subset of automatic question generation. Section 2 commences with an introduction to the necessary concepts for understanding the work in the field. Subsequently, we examine various models addressing the task of visual question generation, categorizing them based on four dimensions: Question Generation Method, Learning Method, Input Modality, and Type of Generated Questions. Our exploration ranges from approaches characterized by precise control over the question generation process, such as rule-based and template-based methods, to those leveraging deep learning to produce creative and unconventional questions, albeit with a trade-off in precise control as they transcend predefined patterns.

In Section 4, we explain the creation of a dataset comprising image-question-answer triples, essential for training the generative models, and show the statistics of the dataset created. We also detail the creation of auxiliary datasets that were used during the training of individual components of the final architecture.

In Section 5, we begin by detailing the publicly available models that we fine-tuned during the development process, as well as the machine learning techniques employed during the training process and the metrics used to evaluate the models. We delve into the architecture where we divide the question

generation task into two tasks and analyze the results of each step until the final model. We examine the scores attributed to the generated questions by a domain specialist (university teacher) and provide our considerations on the generated questions and future work.

In Section 6, we start by explaining the architecture of our solution to the task of question answering, where we divide the question answering task into two tasks and analyze the results of each step until we reach the final model. We provide statistics on the chart types where the model performs better or worse, and we discuss the behavior of the models for each type of chart.

Lastly, in Section 7, we conclude by summarizing the work we developed, providing possible paths for future work, and offering our final considerations on the solutions developed.

# 2

# Literature Review

## Contents



In contrast to other fields, automatic question generation has received relatively less attention, with its practical application being infrequent [26]. Nevertheless, in recent years, advancements in Natural Language Processing (NLP) techniques have introduced more powerful tools, which have found their place in the realm of automatic question generation. Their use in automatically generating sentences has yielded considerable success, enabling the creation of diverse textual question types, including fill-in-the-blank, word formation, multiple-choice, and error correction questions [27], [28].

Visual question generation, a subset of automatic question generation, has witnessed significant strides in computer vision and natural language processing, leading to the development of more robust tools. The application of these advanced techniques to the task of automatically generating questions related to visual content has proven successful. This has empowered the creation of a wide spectrum of questions, ranging from inquiries about objects and scenes to more intricate queries about relationships and contextual details in images or videos, thus enhancing the versatility and interactivity of visual con-

tent understanding and engagement. However, to the best of our knowledge automatically generating visual questions with the purpose of teaching data science has not been addressed before and so our main goal is to automate the generation of questions related to data charts, with a specific emphasis on crafting true or false data science questions.

## 2.1 Basic Concepts

In this section, I will provide brief explanations about concepts that will be necessary to understand the current approaches in the field of visual question generation. The elucidation of these foundational elements not only establishes a common understanding but also lays the groundwork for more nuanced discussions in later chapters. This section, in particular, centers on neural networks and deep learning concepts, given their prevalent usage in the majority of visual question generation approaches.

### 2.1.1 Multilayer Perceptron (MLP)

A Multilayer Perceptron (MLP) [1] is a type of artificial neural network architecture, often considered the simplest type, where data moves in one direction—from the input layer through hidden layers to the output layer. The term ”multilayer” indicates that the network consists of multiple layers of interconnected nodes or artificial neurons see Figure 2.1.

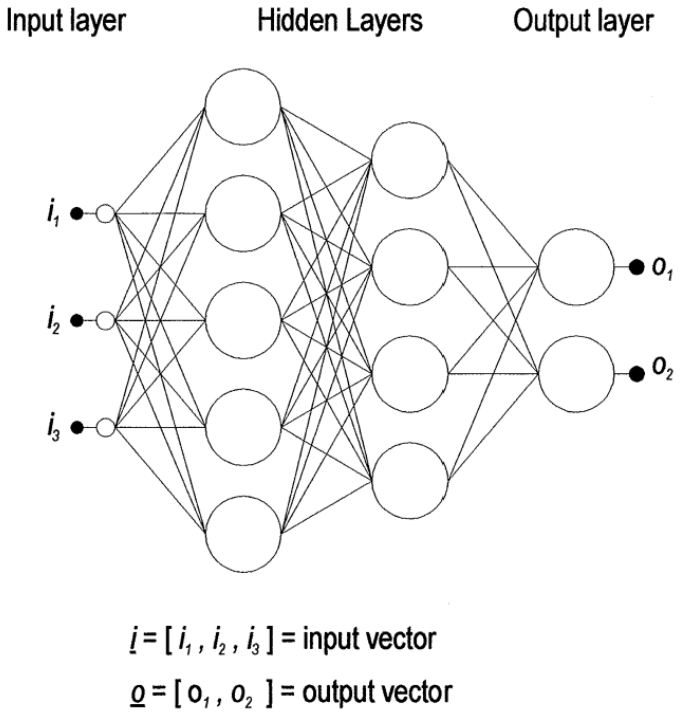



**Figure 2.1:** A multilayer perceptron with two hidden layers. From Gardner et al. [1].

MLPs contain the following key elements:

- An input layer that represents the features or variables of the dataset.
- Hidden layers between the input and output layers, there can be one or more hidden layers. Each node in a hidden layer is connected to every node in the previous and the following layer. These connections have assigned weights that are adjusted during the training process.
- The output layer that produces the final results of the network's computation.
- An activation function, each node, or artificial neuron, in the hidden and output layers typically applies an activation function to the weighted sum of its inputs. Some common activation functions include the sigmoid, hyperbolic tangent (tanh), and rectified linear unit (ReLU). The choice of activation function for the output layer depends on the task at hand. For instance, the sigmoid function is well-suited for binary classification, modeling the probability of an example belonging to one of two classes independently. Conversely, the softmax activation function is tailored for multi-class classification, normalizing output scores into a probability distribution over multiple classes, ensuring the probabilities sum to 1.
- The training process in MLPs employs supervised learning. The model is presented with input-output pairs, and weights are iteratively adjusted to minimize the difference between predicted and actual outputs. This is typically accomplished through backpropagation and optimization algorithms such as gradient descent. The learning process can be outlined as follows:
  - **Initialization**: The neural network is initialized with random weights and biases, determining the strength of connections between neurons.
  - **Forward Pass**: During the forward pass, input data is sequentially fed into the network. Each neuron in a layer receives input, applies a weighted sum, adds a bias, and passes the result through an activation function. This process is repeated until the final output is obtained.
  - **Loss Computation**: The output from the forward pass is compared to the true output (ground truth), and the difference is quantified using a loss function. The choice of the loss function depends on the task, with mean squared error commonly used for regression tasks and cross-entropy for classification tasks (in this scenario, the choice of the activation function for the output layer depends on the nature of the task, employing either softmax for multiclass classification or sigmoid for binary classification).
  - **Backward Pass (Backpropagation)**: The backward pass involves calculating the gradient of the loss with respect to the weights and biases throughout the network, leveraging the chain rule of calculus. The gradients indicate how much the loss would increase or decrease if the weights and biases were adjusted.

– **Parameter Update**: The calculated gradients are used to update the weights and biases in the network. This update is typically performed using an optimization algorithm like stochastic gradient descent (SGD) or one of its variants, adjusting the parameters in the opposite direction of the gradient to minimize the loss.

– **Iteration (Epoch)**: These steps are repeated for multiple iterations, known as epochs, until convergence is achieved.

MLPs are known for their ability to learn complex relationships in data, and their architecture allows them to approximate a wide range of functions. They are the building blocks of many approaches in the context of visual question generation, as we will see ahead in section 3.2.

### 2.1.2 Convolutional Neural Networks (CNN)

Convolutional Neural Networks [2], often abbreviated as CNNs, represent a fundamental component of deep learning, particularly in the domain of computer vision and image analysis. CNNs are a class of artificial neural networks designed to process and analyze visual data, making them highly relevant to a wide range of applications, from image recognition to medical imaging and autonomous vehicles.

At their core, CNNs are inspired by the human visual system. They consist of multiple layers, including convolutional see Figure 2.2, pooling, and fully connected layers, which are specifically designed to recognize patterns and hierarchies of features within an input image. The hallmark of CNNs is their use of convolutional layers, which apply a set of learnable filters to scan and capture local features within an image. These filters move across the image, performing mathematical operations to detect edges, shapes, and other visual elements.

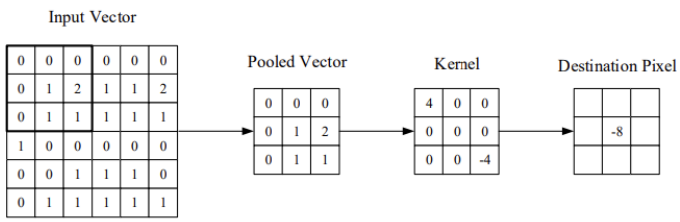



**Figure 2.2:** A visual representation of a convolutional layer. The centre element of the kernel is placed over the input vector, of which is then calculated and replaced with a weighted sum of itself and any nearby pixels. From Nash et al. [2].

A critical advantage of CNNs over MLPs is their ability to automatically learn and adapt to features in visual data. Using backpropagation, the network fine-tunes its parameters during training, optimizing its ability to identify complex patterns and objects. This adaptability is particularly useful for tasks like object recognition, where variations in size, orientation, and lighting must be accommodated [29].

The use of CNNs has yielded remarkable results in numerous domains, revolutionizing image analysis, and enabling breakthroughs in fields like facial recognition [30], object detection [31], and medical image diagnosis [32]. CNNs are used in almost every approach in the context of visual question generation since they provide an excellent way of processing visual data.

### 2.1.3 Recurrent Neural Network (RNN)

A Recurrent Neural Network (RNN) [33] is a type of artificial neural network designed to process sequential data by maintaining a hidden state that captures information about previous elements in the sequence. Unlike feedforward neural networks, where data flows in a single direction, from input to output, RNNs have connections that form directed cycles, allowing them to exhibit temporal dynamic behavior. This in turn allows RNNs to deal with sequences while traditional fully connected feedforward networks can only deal with input vectors.

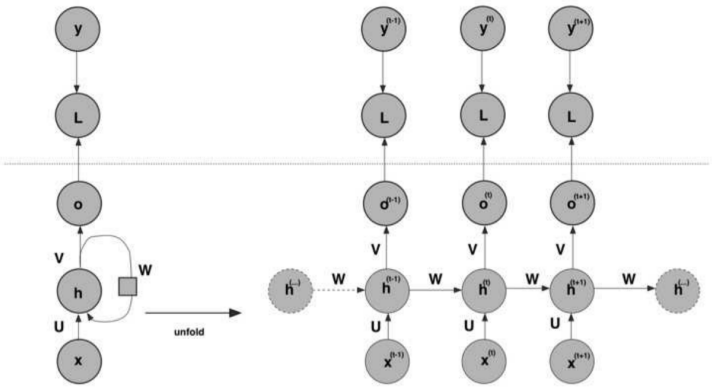



**Figure 2.3:** RNN architecture. The outputs are indicated by $o^{(t)}$ and the target values by $y^{(t)}$, the costs are represented by a loss function L. From Wichert et al. [3].

Figure 2.3 Illustrates the architecture of a Recurrent Neural Network (RNN). Establishing connections between hidden layers at sequential timesteps requires the use of connection weights. These weights, denoted by a squared matrix $W$, are shared across all connections between pairs of hidden states. The relationship can be expressed as follows:

$$h^{(t)} = \phi(U \cdot x^{(t)} + b + W \cdot h^{(t-1)}) \tag{2.1}$$

In this equation, $h^{(t)}$ represents the hidden activations at time $t$, which depend on the input at that time $x^{(t)}$ as well as the previous hidden state $h^{(t-1)}$. Similar to the previously described Multi-Layer Perceptron (MLP), an activation function $\phi$ is applied.

Although any activation function $\phi$ can be employed, the hyperbolic tangent (tanh) is commonly used

in RNNs. Tanh helps address the vanishing gradient problem by mapping input values to a range where gradients are less likely to vanish during backpropagation through time. Unlike activation functions like the sigmoid (commonly used in the early days of RNNs), which map values to the range [0, 1], tanh's output range is centered around zero within the range of [-1, 1]. This characteristic allows tanh to effectively capture both positive and negative signals, aiding in the preservation of gradient signs during backpropagation.

While RNNs are effective in modeling sequential dependencies, they face challenges with long-term dependencies and suffer from the vanishing gradient problem. As a result, more advanced RNN variants, such as Long Short-Term Memory (LSTM) networks, have been developed to address these issues and improve the learning of long-range dependencies.

### 2.1.4 Long short-term memory (LSTM)

LSTM [34] is a type of recurrent neural network (RNN) architecture that has gained significant prominence due to its unique ability to effectively model and process sequences of data. Traditional RNNs are limited by their vanishing gradient problem, which hampers their capacity to capture long-range dependencies in sequences. In contrast, LSTM networks were explicitly designed to address this limitation by introducing specialized memory cells and gating mechanisms.

At the core of LSTM are memory cells that can store and retrieve information over extended sequences. These cells are controlled by gating mechanisms, including the input gate, forget gate, and output gate. These gates regulate the flow of data into and out of the memory cells, enabling LSTMs to manage and retain valuable information over time Figure 2.4.

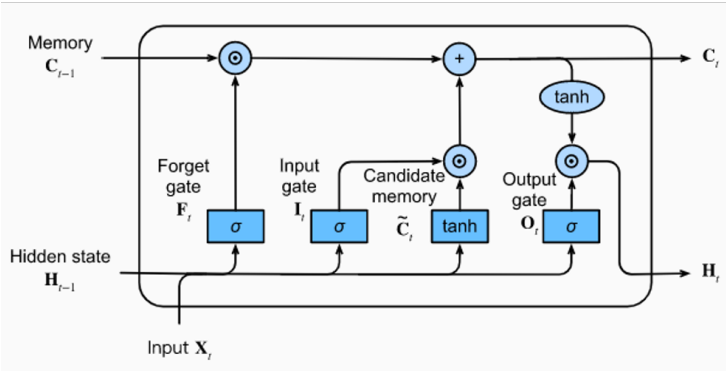



**Figure 2.4:** Architecture of a LSTM Unit (Credits: [4]).

- Input Gate: The input gate determines which information is relevant to store in the memory cell at a given time step. It selectively updates the cell state with new information from the current input.
- Forget Gate: The forget gate decides which information in the memory cell should be discarded or

forgotten. It prevents the network from retaining irrelevant or outdated data.

- Output Gate: The output gate controls what information from the memory cell is used to generate the output at a specific time step. It ensures that the LSTM produces meaningful predictions or representations.

By incorporating these components, LSTMs can effectively capture long-range dependencies in sequential data, making them well-suited for tasks such as natural language processing, speech recognition, and time series prediction. In the context of visual question generation, they are often used to decode the output to readable questions.

### 2.1.5 Encoder-Decoder architecture

The encoder-decoder architecture [35] is a neural network design pattern commonly used for sequence-to-sequence tasks, where the input and output are both sequences of varying lengths. This architecture consists of two main components: an encoder and a decoder. Each component is designed to handle a different aspect of the sequence transformation task see Figure 2.5.


**Figure 2.5:** The encoder–decoder architecture (Credits: [5]).

The encoder processes the input sequence and produces a fixed-size context representation or a set of hidden states that capture the essential information from the input. This context representation serves as a condensed and informative summary of the input sequence. Commonly used architectures for the encoder include recurrent neural networks (RNNs), Long Short-Term Memory networks (LSTMs), or Transformer networks.

The decoder takes the context representation produced by the encoder and generates the output sequence. It operates in a autoregressive manner, where it generates one element at a time based on the context representation and the previously generated elements. The decoder can be implemented using RNNs, LSTMs, Transformers, or other architectures suitable for sequence generation.

In the context of visual question generation the encoder might use CNNs since we receive images as input and for the decoder it is possible to use RNNs, LSTMs, Transformers, or other architectures suitable for sequence generation as we will see in the literature review section.

### 2.1.6 Attention Mechanism

Attention [36] is an Encoder-Decoder kind of neural network architecture that refers to a mechanism that allows a model to focus on specific parts of the input sequence when making predictions or decisions. This concept is inspired by how human attention works when, for example, reading a paragraph or observing an image.

In neural networks, attention mechanisms are particularly useful in sequence-to-sequence tasks, such as machine translation or text summarization, and in computer vision tasks, like visual question generation and image captioning. The key idea is to dynamically allocate different levels of importance (weights) to different parts of the input sequence, rather than treating all parts equally.

The general attention mechanism (Figure 2.6) incorporates three main components: queries (Q), keys (K), and values (V). The mechanism follows these computations:

- Each query vector, $q = s_{t-1}$ , is matched against a database of keys to compute a score value. This matching operation is computed as the dot product of the specific query under consideration with each key vector, $k_i$:

$$e_{q,k_i} = q \cdot k_i \tag{2.2}$$

- The scores then pass through a softmax operation to generate weights:

$$\alpha_{q,k_i} = softmax(e_{q,k_i}) \tag{2.3}$$

- The generalized attention is then computed by a weighted sum of the value vectors, $v_{k_i}$, pairing each value vector with its corresponding key:

$$attention(q, K, V) = \sum_i \alpha_{q,k_i} v_{k_i} \tag{2.4}$$

In essence, when presented with a sequence of words, the generalized attention mechanism takes the query vector associated with a specific word and scores it against each key in the database. This process captures how the word relates to others in the sequence. The values are then scaled according to attention weights, retaining focus on words relevant to the query and producing an attention output for the word under consideration.

Attention mechanisms enable models to emphasize relevant information while disregarding irrelevant details, thereby enhancing performance on tasks that involve handling long sequences and capturing dependencies between distant elements in the input.

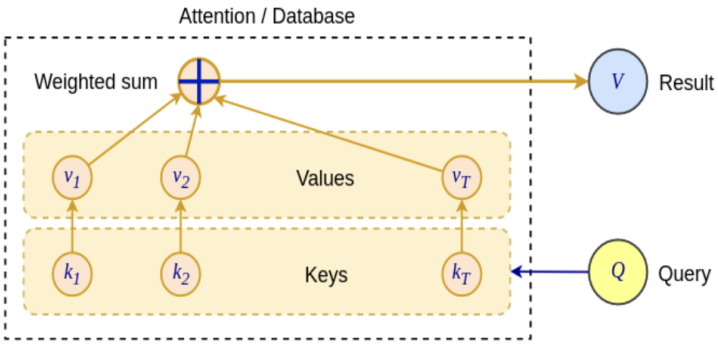



**Figure 2.6:** Diagram illustrating general attention (Credits: [6]).

### 2.1.7 The Transformer Architecture

The transformer architecture is a type of neural network architecture introduced in the paper ”Attention is All You Need” by Vaswani et al. [7] The transformer architecture has become a fundamental building block for various natural language processing (NLP) tasks, such as machine translation, text summarization, and question answering.

The invention of the attention mechanism solved the problem of how to encode a context into a word, or in other words, how you can present a word and its context together in a numerical vector. The transformer takes this concept a step further, constructing a neural network for natural language translation without recurrent structures. This simplifies the network, making it more trainable, parallelizable, and capable of accommodating complex language models.

The creation of transformers was fueled by the need to overcome the limitations of traditional RNNs. Unlike RNNs, which process sequences sequentially, thereby restricting parallelization and potentially leading to slower training times, transformers introduce the self-attention mechanism. This innovation empowers transformers to capture dependencies between different positions in the input sequence concurrently, facilitating parallel processing. This parallelization not only accelerates training but also enhances the efficiency of transformers in effectively managing long-range dependencies within sequences.

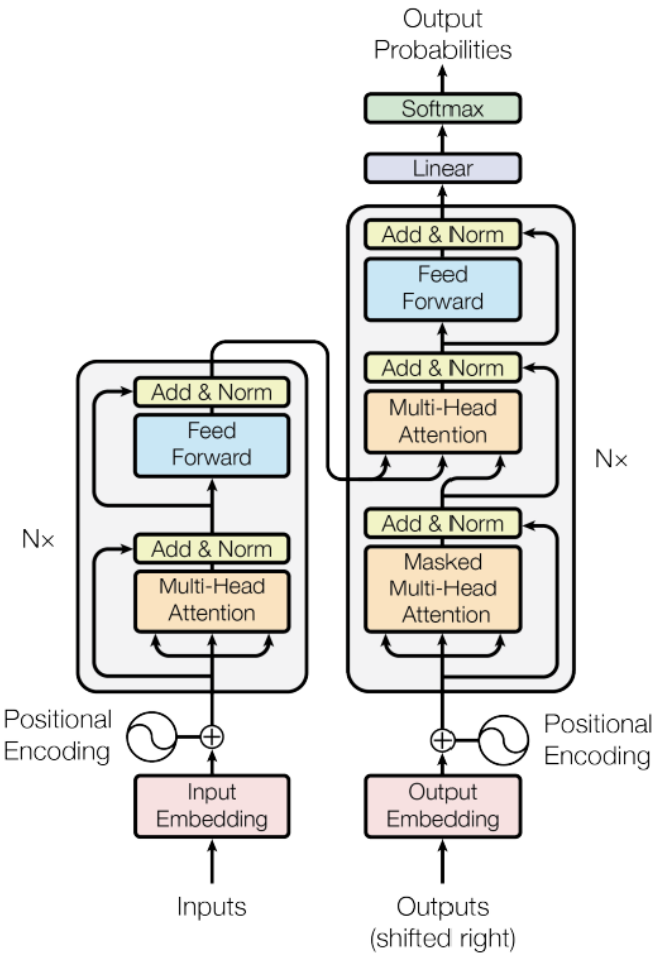



**Figure 2.7:** The Transformer - model architecture. From Vaswani et al. [7].

In Figure 2.7 we can see that the Transformer architecture follows an encoder-decoder structure but does not rely on recurrence and convolutions in order to generate an output.

The task of the encoder, on the left half of the Transformer architecture, is to map an input sequence to a sequence of continuous representations, which is then fed into a decoder. To be more precise, the encoder consists of a stack of $N = 6$ identical layers, where each layer is composed of two sublayers:

- The first sublayer implements a multi-head self-attention mechanism. This mechanism implements $h$ heads that receive a (different) linearly projected version of the queries, keys, and values, each to produce $h$ outputs in parallel that are then used to generate a final result.

- The second sublayer is a fully connected feed-forward network consisting of two linear transformations with Rectified Linear Unit (ReLU) activation in between:

$$FFN(x) = ReLU(W_1 x + b_1)W_2 + b_2 \tag{2.5}$$

Each sublayer incorporates a residual connection and is succeeded by a normalization layer, $layernorm(.)$,which normalizes the sum computed between the sublayer input, $x$, and the output generated by the sublayer itself, $sublayer(x)$:

$$layernorm(x + sublayer(x)) \tag{2.6}$$

Since it does not make use of recurrence, the Transformer architecture cannot inherently capture any information about the relative positions of the words in the sequence. So this information has to be injected by introducing positional encodings to the input embeddings.

The positional encoding vectors, matching the dimension of input embeddings, are generated through the application of sine and cosine functions at various frequencies. Subsequently, these vectors are simply summed to the input embeddings, thereby introducing positional information into the data.

The decoder, on the right side, processes the encoder's output alongside the decoder's previous output to generate a sequence. Comprising $N = 6$ identical layers, each decoder layer consists of three sublayers:

- The initial sublayer incorporates the prior output from the decoder stack, augments it with positional information, and employs a multi-head self-attention mechanism. Unlike the encoder, which attends to all words in the input sequence irrespective of their position, the decoder is adjusted to focus solely on preceding words. This restriction ensures that the prediction for a word at position $i$ relies solely on the known outputs for preceding words in the sequence. In the multi-head attention mechanism (which implements multiple, single attention functions in parallel), this is achieved by introducing a mask over the values produced by the scaled multiplication of matrices $Q$ and $K$. This masking is implemented by suppressing the matrix values that would otherwise correspond to illegal connections:

$$mask(QK^T) = mask(\begin{bmatrix} e_{11} & e_{12} & \dots & e_{1n} \\ e_{21} & e_{22} & \dots & e_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ e_{m1} & e_{m2} & \dots & e_{mn} \end{bmatrix}) = \begin{bmatrix} e_{11} & -\infty & \dots & -\infty \\ e_{21} & e_{22} & \dots & -\infty \\ \vdots & \vdots & \ddots & \vdots \\ e_{m1} & e_{m2} & \dots & e_{mn} \end{bmatrix} \tag{2.7}$$

- The second layer implements a multi-head self-attention mechanism that mirrors the one employed in the encoder's first sublayer. On the decoder side, this multi-head mechanism receives queries from the preceding decoder sublayer and keys and values from the encoder's output, enabling the decoder to attend to all words in the input sequence.
- The third layer implements a fully connected feed-forward network, similar to the one implemented in the second sublayer of the encoder.

Residual connections and normalization layers follow each sublayer. Positional encodings are added to the decoder's input embeddings, mirroring the process in the encoder.

### 2.1.8 Large Language Models

Large Language Models (LLMs) are advanced natural language processing models that are characterized by their extensive scale, capacity, and ability to understand and generate human-like text. These models are trained on vast amounts of textual data and use sophisticated architectures to capture and generate language patterns.

They often use transformer architectures (Figure 2.7) as a foundational component of their design. The transformer architecture, as mentioned before, has become a standard for processing sequential data, especially in natural language processing tasks.

Large Language Models like GPT-3 (Generative Pre-trained Transformer 3) [37] and BERT (Bidirectional Encoder Representations from Transformers) are examples of LLMs that are built on top of transformer architectures. These models leverage the transformer's ability to capture contextual information and handle sequential data efficiently, enabling them to achieve state-of-the-art performance on various natural language processing tasks. Key characteristics of these models include:

- **Scale:** These models are trained on vast datasets, often consisting of terabytes of text data. The large-scale training enables them to capture a wide range of linguistic patterns, semantics, and contextual information.
- **Parameter Count:** Large language models have a massive number of parameters (weights) in the order of billions, allowing them to learn complex relationships within the data.
- **Transfer Learning:** They leverage transfer learning, where the models are pretrained on a general language task (like predicting the next word in a sentence) and then fine-tuned for specific downstream tasks, making them versatile across various applications.
- **Contextual Understanding:** Due to their scale, these models exhibit a strong contextual understanding of language. They can generate coherent and contextually relevant text, answer questions, and perform language-related tasks.

These models have found applications in a variety of NLP tasks, including text generation, sentiment analysis, summarization, translation, question answering, and question generation.

### 2.1.9 Variational Autoencoders

Variational Autoencoders (VAEs) [38] are a type of generative model in the field of machine learning. They belong to the family of autoencoders but introduce a probabilistic approach to the encoding and decoding process. VAEs are particularly useful for generating new data points that resemble a given dataset.

Instead of outputting the vectors in the latent space like in normal autoencoders, the encoder of VAE outputs parameters of a pre-defined distribution in the latent space for every input. The VAE then imposes a constraint on this latent distribution forcing it to be a normal distribution. This constraint makes sure that the latent space is regularized.

Integrating VAEs into visual question generation introduces a probabilistic element to the process, enabling the model to capture uncertainty and generate diverse questions for a given visual input.

### 2.1.10 Generative Adversarial Networks

Generative Adversarial Networks (GANs) are a class of artificial intelligence algorithms used in unsupervised machine learning. They were introduced by Ian Goodfellow et al. [39] and consist of two neural networks Figure 2.8, a generator G, and a discriminator D, which are trained simultaneously through adversarial training.

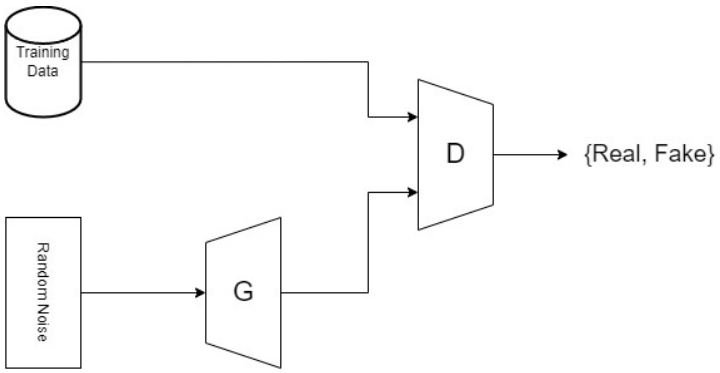



**Figure 2.8:** The GAN architecture.

GANs work by having a generator G that takes random noise as input and tries to generate synthetic data that resembles real data from a training dataset. The discriminator D, on the other hand, is trained to distinguish between real data from the training set and the synthetic data created by the generator. The generator and discriminator are trained simultaneously in a competitive manner. The generator aims to produce data that is indistinguishable from real data, while the discriminator aims to correctly classify whether the input data is real or generated.

The objective is for the generator to improve over time so that it generates data that is increasingly difficult for the discriminator to differentiate from real data. Conversely, the discriminator aims to become more accurate in distinguishing between real and generated data. Ideally, this adversarial process reaches an equilibrium where the generator produces high-quality synthetic data, and the discriminator is unable to distinguish it from real data.

GANs have found applications in various domains and in the context of visual question generation they have been used in some approaches as we will see in the next section.

## 2.2 Visual Question Generation

Visual question generation VQG is a relatively new problem as compared to visual question answering VQA. Both are born from the intersection of computer vision and natural language processing but the emphasis in VQA is on image understanding, question understanding, and finding out the answer using image and question correlation. VQG aims at generating multiple, diverse, unique, unambiguous, reasonable, and syntactically correct questions after understanding the image [40]. Although much less work is been done in this area, we will analyze the efforts done so far to tackle this problem.

Visual Question Generation represents a dynamic convergence of various elements, each influencing the process and the quality of generated questions. In this section, we delve deeper into four pivotal dimensions inspired by the categorization used in the review by Patil et al. [40]:

- Question Generation Method: We analyze the methods employed for generating questions, ranging from traditional rule-based and template-based approaches to novel question generation techniques. Each method carries distinct advantages and limitations, and understanding their nuances is paramount for comprehending the evolution of VQG algorithms.
- Learning Method: The choice of learning methods plays a pivotal role in the development of VQG models. We explore the spectrum from simple deep neural networks to more advanced generative models and reinforcement learning networks.
- Input Modality: The information fed into VQG models significantly influences their performance. We investigate different input modalities, including image-only, image with contextual information, and image paired with answer data.
- Type of Generated Questions: VQG algorithms produce an array of question types, spanning from questions grounded entirely in visual content to those that draw upon common sense or external knowledge.

### 2.2.1 Question Generation Method

#### A – Rule Based

Rule-based visual question generation relies on predefined rules to create questions that are contextually pertinent to the visual content. This method typically involves parsing the relevant text, such as image descriptions, and applying specific syntactic rules to the parsed structure in order to generate questions.

The Visual Question Generation (VQG) approach implemented in COCO QA [41] exemplifies a rule-based synthesis of image labeling. In this approach, the authors used the MS-COCO dataset [42], which serves as an image description dataset. They employed the Stanford parser [43] to obtain the syntactic

structure of the original image descriptions. Subsequently, the algorithm adhered to certain rules to construct questions:

- It divides compound sentences into two simpler sentences. Compound sentences, consisting of two independent clauses connected by a conjunctive word, are split into two distinct sentences.
- It transforms indefinite determiners such as ”a(n)” into definite determiners like ”the”.
- The algorithm repositions the verb and the ”wh-” constituent to the beginning of the sentence. For instance, ”A woman is driving a car” is transformed into ”What is the woman driving?”.

The authors generated four categories of questions:

- Object Questions: These pertain to inquiries about objects using the word ”what.” The original sentence is modified by replacing the actual object with ”what” and altering the sentence structure accordingly. The process follows the aforementioned rules: (1) Breaking down complex sentences into simpler ones; (2) Substituting indefinite determiners with definite ones; (3) Identifying potential answers and replacing them with ”what.”
- Number Questions: These questions follow a similar procedure as the Object Questions. The distinction lies in how potential answers are identified; numbers are extracted from the original sentences. Nevertheless, the steps involving sentence simplification, determiner modification, and wh-movement remain consistent.
- Color Questions: Generating color-related questions involves identifying the color adjective and the noun to which the adjective pertains. A sentence is then structured as ”What is the color of the [object],” with the ”object” being replaced by the specific noun.
- Location Questions: These questions are akin to Object Questions, with the primary difference being that answer extraction is restricted to constituents that commence with the preposition ”in.”

Rule-based visual question generation is advantageous in scenarios demanding precise control over question generation and when the relationships between visual elements, in this case, represented through image descriptions, and questions are clearly defined. However, it lacks the flexibility and adaptability offered by machine learning-based approaches, which have the capacity to discern patterns and generate questions based on a broader range of data.

**B – Template Based**

Template-based approaches pre-define the syntactic structures for question construction. This method streamlines the question generation process by using fixed question formats that contain designated slots for incorporating specific visual elements. A question is completed by filling appropriate

positions in these structures using relevant objects, their attributes, and actions identified in the input image.

There are two popular approaches. In one approach, a graph memory representation of the visual concepts within the image is leveraged to fill the gaps in question templates. An illustrative example of this approach is the visual curiosity model [8], which employs a template-based methodology for generating questions based on recognized visual concepts.

To elaborate further, the authors introduce a novel framework for learning visual curiosity, formulating it as a reinforcement learning problem. They devise an agent equipped with visual curiosity, enabling it to pose questions to an Oracle (e.g., a human) regarding image contents (e.g., 'What is the object on the left side of the red cube?') and construct a visual recognition model based on the responses received (e.g., 'Cylinder').

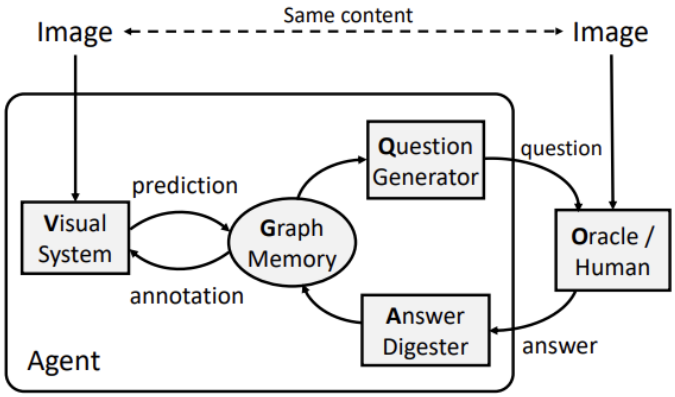



**Figure 2.9:** The visual curiosity model framework. From Yang et al. [8].

Of particular relevance in the context of visual question generation is the Question Generator module (Figure 2.9), responsible for identifying object proposals to inquire about and generating questions based on the graph memory to query the Oracle about object categories or attributes. The graph memory is depicted as a directed graph (G = (V, E)) associated with an image I, with nodes (V) corresponding to object proposals and edges (E) representing relationships between proposals.

To produce informative questions, the agent selects from a set of question templates, filling in information retrieved from the graph memory. This process essentially involves determining the target attribute, target object, and reference object to inquire about. For example, a generated question might be "What is the <white> <object> besides the <red object>." Here, the target object and attribute are '<object>' and '<white>', and the reference object is '<red object>'.

The authors implement the question generation policy by employing a Recurrent Neural Network (RNN) to track which questions have been asked and whether they were meaningful or not according to the responses from Oracle (i.e. referring to valid objects).

In the alternative approach, all questions are generated using a functional programming method that harnesses the scene graph of the input image. This approach involves extracting grounded facts from the image and employing templates to construct complex questions requiring systematic reasoning for answers. The GQA dataset [9] is an example of this approach, created by harnessing the Visual Genome scene graph structures [44] to generate diverse reasoning questions.

The question generation process in this approach draws upon four key resources: the scene graphs from Visual Genome Scene Graph annotations, offering rich content concerning objects, attributes, and relationships; structural patterns or templates, shaping the content into questions; references, used to point to objects in the image; and decoys, facilitating the generation of negative questions or those involving logical inference.

The system operates based on question groups, where each group is associated with (1) a functional program representing its semantics; (2) a set of textual rephrases conveying the question in natural language (e.g., "What|Which <type> [do you think]?"); and (3) a pair of short (e.g., "<attribute>") and long answers (e.g., "The <object> <is> <attribute>"), encapsulating the structural pattern.

For each object, the system computes a range of candidate references, both direct and indirect. Direct references, such as 'the bear' or 'this animal,' are used when object uniqueness is assured by object detectors, making the references unambiguous. Indirect references involve modifiers, leading to multi-step questions as varied as "Who is looking at the animal that is wearing the red coat in front of the window?" and thus increasing question flexibility. This is the key ingredient which leads to questions that require reasoning.

Furthermore, the system computes a set of decoys for scene graph elements, especially for negative questions or those involving logical inference. These decoys are selected based on their likelihood to be related to the subject and their plausibility in the context of other objects in the scene.

**A1**. Is the **tray** on top of the **table** black or light brown? light brown
**A2**. Are the **napkin** and the **cup** the same color? yes
**A3**. Is the small **table** both oval and wooden? yes
**A4**. Is there any **fruit** to the left of the **tray** the **cup** is on top of? yes
**A5**. Are there any **cups** to the left of the **tray** on top of the **table**? no
**B1**. What is the brown **animal** sitting inside of? **box**
**B2**. What is the large **container** made of? cardboard
**B3**. What **animal** is in the **box**? **bear**
**B4**. Is there a **bag** to the right of the green **door**? no
**B5**. Is there a **box** inside the plastic **bag**? no

**Figure 2.10:** Examples of questions from the GQA dataset. From Hudson et al. [9].

In the final step, the system traverses the graph and generates pertinent questions by instantiating a randomly selected question pattern. For example, "What <type> is <theObject>, <attribute> or <cAttribute>?," where all fields are populated with matching information, leading to questions like "What (color) (is) the (apple on the table), (red) or (green)?".

The main difference between the two approaches lies in the fact that, in the first one, only the graph memory representation of visual concepts in the image is employed to fill the gaps in the question templates, while in this one the templates are used to generate complex questions that need systematic reasoning to answer.

Template-based question generation offers advantages such as simplicity of implementation, generation of consistent questions, and precise control over the generated content. However, it also has limitations in terms of flexibility and adaptability when compared to machine learning-based approaches.

### C – Novel Question Generation

Novel visual question generation places a significant emphasis on the generation of creative and distinct questions that transcend conventional or predefined patterns. In contrast to rule-based or template-driven approaches, novel visual question generation seeks to produce inquiries that are inventive, contextually relevant, and free from predictability. This endeavor typically entails the use of machine learning models, particularly deep neural networks, to dynamically generate questions based on the visual content.

One notable illustration of novel visual question generation, particularly pertinent in this context, is the iQAN model [10]. The Inverse Visual Question Answering (IVQA) task involves crafting questions for an image centered around a specific answer. In their research, the authors undertook the challenge

of visual question generation as a dual task intertwined with visual question answering. Within this framework, answer-based visual question generation, referred to as VQG for simplicity, is regarded as an inverse variant of Visual Question Answering (VQA) Figure 2.11. It entails the generation of a question that corresponds to a provided image-answer pair. The VQG model merges the answer embedding $a$ and the image feature $v$ to derive the question embedding $\hat{q}$, which is subsequently decoded to produce the question sentence.

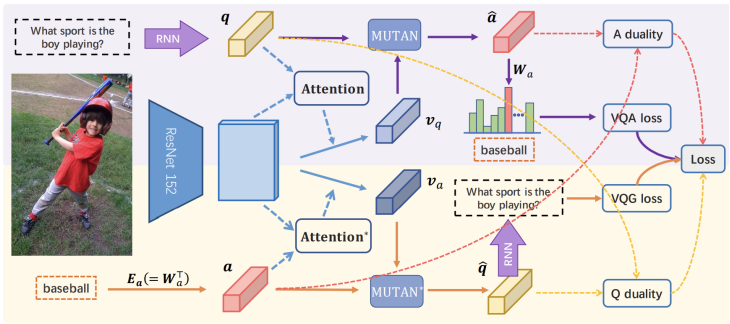



**Figure 2.11:** Overview of Invertible Question Answering Network (iQAN), which consists two components for VQA and VQG respectively. The upper component is MUTAN VQA component, and the lower component is its dual VQG model. From Li et al. [10].

In the initial stages, the authors harnessed the ResNet-152 architecture [45] to extract image features. ResNet-152 is a specialized deep convolutional neural network architecture characterized by the use of residual connections that facilitate the training of very deep neural networks while mitigating the vanishing gradient problem that frequently impedes the training of such deep architectures.

Within the VQA component, an RNN is employed to obtain the embedded feature $q$ from the input question, while a CNN transforms the input image into a feature map $v$. An attention module based on MUTAN is used to generate a question-aware visual feature $v_q$ by incorporating information from both the image and the question. MUTAN, in this context, takes as input the image feature $v_q$ and the question feature $q$ to predict the answer feature $\hat{a}$. Subsequently, another MUTAN fusion module is employed to obtain the answer features $\hat{a}$ by merging $v_q$ and $q$. Finally, a linear classifier $W_a$ is used to predict the answer.

In the VQG or IVQA component, when provided with an answer, a lookup table $E_a$ containing answer embeddings is used to obtain the embedded feature $a$. A CNN equipped with an attention module is instrumental in extracting the visual feature $v_a$ from the input image and the answer feature $a$. The dual-form MUTAN, sharing parameters with the VQA MUTAN but structured differently, is then deployed to acquire the predicted question features $\hat{q}$. These features are subsequently decoded using an LSTM-based decoder to transform $\hat{q}$ into the question sentence.

By employing this model (Figure 2.11), the authors are equipped to generate questions that are not confined by templates or rigid rules, resulting in the production of creative and unique questions. The

generative and reinforcement learning models are also capable of generating novel questions. They are explained in the next section.

### 2.2.2 Learning Method

#### A – Deep Neural Networks

The majority of the proposed models in the field of Visual Question Generation leverage deep neural networks to facilitate question generation, showcasing their capability to generate novel and contextually relevant inquiries. One such exemplar is the IQAN model [10], discussed in the preceding section. The IQAN model uses ResNet-152, a convolutional neural network architecture, with attention mechanisms to extract essential visual features. Furthermore, it incorporates MUTAN, a neural network architecture specially tailored to predict question features. Finally, the approach culminates in the use of an LSTM-based decoder to transform the question features into coherent and informative question sentences.

Another noteworthy model is the Multimodal Differential Network [11]. This algorithm can be dissected into three core modules Figure 2.12.

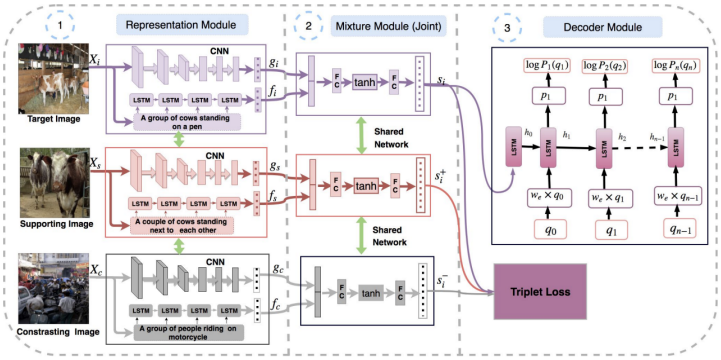



**Figure 2.12:** This is an overview of our Multimodal Differential Network for Visual Question Generation. It consists of a Representation Module which extracts multimodal features, a Mixture Module that fuses the multimodal representation and a Decoder that generates question using an LSTM based language model. From Patro et al. [11].

The Representation Module takes as its input three images and their respective captions: the target image, representing the central visual content around which questions revolve; the supporting image, an auxiliary image providing contextual information to assist in question generation; and the contrasting image, which introduces diversity and variation into the question generation process. The Representation Module obtains visual embeddings through a Convolutional Neural Network (CNN) and caption embeddings through a Long Short-Term Memory (LSTM) network. Subsequently, these embeddings are transferred to the Mixture Module.

Within the Mixture Module, a fusion of these multimodal embeddings takes place, where they are

concatenated and mapped to a fixed-length vector. This vector then passed through an LSTM-based decoder, generating contextually relevant questions.

### B – Generative Networks

Generative networks, often referred to as generative models, are a category of artificial neural networks designed to produce data, commonly in the forms of images, text, or structured information. Their primary purpose is to create new data instances that resemble, but aren't identical to, their training data. These networks have diverse applications spanning image generation, natural language processing, and data synthesis. With these generative capabilities in mind, various models have leveraged generative networks.

One notable approach, introduced by Krishna et al. [12], is based on Variational Autoencoders (VAEs) and focuses on generating purpose-driven questions. This model's objective is to generate questions tailored to a given image and answer category pair. Image and answer embeddings are obtained using CNN and LSTM, similar to previous models. Answers are categorized into predefined categories (e.g., objects, attributes, color) represented as one-hot vectors $h_c$ and combined with image embeddings $h_i$ in the variational t-space.

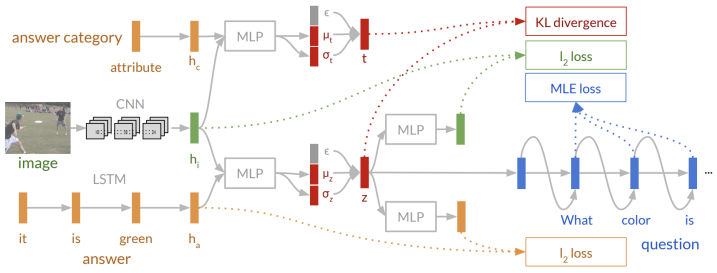



**Figure 2.13:** The image and answer are embedded into a latent space z and an attempt is made to reconstruct them, thereby maximizing mutual information with the image and the answer. z is used to generate questions and train with an MLE objective. Finally,a second latent space t that is trained by minimizing KL-divergence with z is introduced. t allows to remove the dependence on the answer when generating questions and instead grants the ability to generate questions conditioned on the answer category. From Krishna et al. [12].

What sets this model (Figure 2.13) apart is its use of two latent spaces. One space maximizes mutual information among the image, expected answer, and generated question, while the second space, learned concurrently, clusters similar answers. The latter space serves as a regularization mechanism for the former. This second latent space is crucial as the model's goal is to generate questions solely based on the image and answer category, allowing it to produce questions when the actual answer is unknown during inference.

In a recent approach by Zhu, He, et al. [13], a generative pre-trained transformer (GPT) model, a type of transformer model, is harnessed to generate higher-quality questions from medical images. This

model aids the diagnostic process and reduces reliance on physician involvement, optimizing medical resource use.

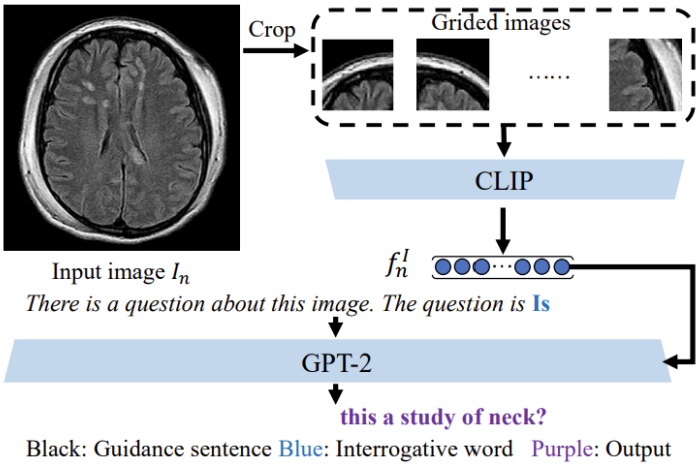



**Figure 2.14:** The model architecture. From Zhu et al. [13].

The proposed Visual Question Generation (VQG) model, depicted in Figure 2.14, integrates CLIP [46], as an image encoder to extract features from grided images and employs GPT-2 for final question generation.

CLIP, is a state-of-the-art machine learning model developed by OpenAI. It was introduced to address the challenge of creating a single model capable of understanding both images and text in a way that enables cross-modal understanding. CLIP is designed to learn a unified representation for images and their corresponding textual descriptions. To do this CLIP maps images and corresponding text into a shared embedding space where the similarity between the representations aligns with semantic similarity. In this shared space, similar images and texts are closer together, facilitating cross-modal understanding.

The model depicted in Figure 2.14 extracts image features from a segmented image grid using the CLIP model, replacing self-attention with cross-attention in GPT-2. This integration introduces large-scale multimodal models into the question generation process. To bootstrap the final output, a structure is designed: Guidance Sentence + Interrogative Word + Output Question. ”There is a question about this image. The question is” serves as the guidance sentence to denote the generation process. Interrogative words are used to specify question categories, enhancing control over the generated questions.

### C – Reinforcement Learning Networks

Visual question generation presents a complex task, demanding an understanding of image content to craft meaningful questions. Reinforcement learning adds a unique dimension by enabling the model to refine its question generation approach through interactions with its environment.

In pursuit of more human-like and natural question generation, Fan et al. [14] employ a two-discriminator strategy. This approach involves a GAN-based dynamic discriminator that distinguishes between human-generated and machine-generated questions, and a static discriminator that assesses question naturalness.

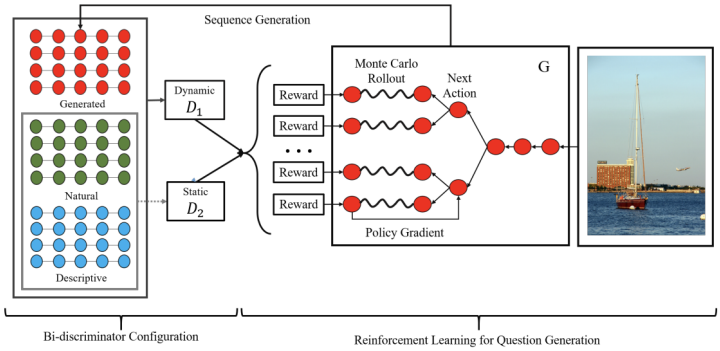



**Figure 2.15:** The overall framework of the model. From Fan et al. [14].

Specifically, the authors propose a model (Figure 2.15) that maximizes two question attributes: ”human-written” (human-written vs. machine-generated) and ”natural” (natural vs. descriptive). To achieve this, they establish a dynamic GAN-based discriminator tasked with discerning human-written from machine-generated questions. It guides the generator to produce questions akin to those originating from humans. The discriminator's training employs human-generated questions as positive samples and generator-generated questions as negatives. The ideal outcome is a seamless blend where the discriminator cannot differentiate between the two question types, assigning equal probabilities to both. This dynamic discriminator is updated upon the arrival of each new batch of generated questions during training.

Simultaneously, a static discriminator is introduced to distinguish natural questions from descriptive ones. It is used to guide the generator to produce questions with information beyond the image to meet the attribute of natural. The static discriminator is pretrained using human-generated samples from both descriptive and natural question domains and remains static during the generator's training.

The optimization of the question generator relies on rewards from these two discriminators. The reinforcement learning algorithm unfolds in four steps: first, the dynamic GAN discriminator update; second, computation of rewards from the GAN discriminator; third, computation of rewards for the static discriminator; and fourth, generator update through Monte Carlo rollout. This approach leverages discriminators to enhance the quality and naturalness of generated questions while using reinforcement learning to iteratively refine the question generation process.

### 2.2.3 Input

#### A – Only Image

Typically, VQG methods that solely rely on an image as input treat the neural network as a black box. The GRNN model [47], exemplifies this approach by exclusively using the image input. On the other hand, the visual curiosity model [8], outlined in the template section, represents a more intricate model that operates solely on images, combining template-based and reinforcement learning techniques.

#### B – Image + Context

Certain VQG approaches incorporate image context, specifically image descriptions, to facilitate question generation. The previously mentioned Multimodal Differential Network [11] serves as an instance of a model that creates captions based on three image and description pairs: the target image, the supporting image, and the contrasting image.

#### C – Image + Answer

This approach establishes a complementary relationship between VQG and VQA. An illustration of this approach is the invertible question answering network (iQAN) [10] discussed previously. The authors embraced the challenge of visual question generation, integrating it as a dual task intricately connected with visual question answering. In this framework, answer-based visual question generation, termed VQG for simplicity, is considered an inverse form of Visual Question Answering (VQA). It involves creating a question that corresponds to a given image-answer pair. Krishna et al. [12], as previously discussed, incorporate the answer category as an input in addition to the image, extending beyond the constraint of requiring an answer for question generation. The second latent space, formed by the answer category and image features during training, enables the acceptance of the answer category alongside the image. This facilitates the generation of questions aimed at maximizing mutual information among the image, answer, and the generated question.

### 2.2.4 Type of generated questions

#### A – Totally grounded

Totally grounded questions rely exclusively on image information for both questions and answers, without the need for external knowledge. For instance, the iQAN model [10] generates questions of this kind, taking an image and an answer as input. In the template section, the functional programming approach discussed in Hudson and Manning [9] uses ground-truth scene graphs to create totally grounded questions using predefined templates.

**B – Natural**

Natural question generation involves producing more human-like, abstract questions based on an understanding of the image beyond literal descriptions. The two-discriminator strategy by Fan et al. [14] exemplifies this approach, striving to maximize two attributes: ”human-written” (human-written vs. machine-generated) and ”natural” (natural vs. descriptive) questions. These kinds of questions are also grounded with the question being formed using a larger vocabulary and understanding the image on an abstract level.

**C – Commonsense-Based**

Commonsense-based questions demand external commonsense knowledge in addition to image-grounded information for both questions and answers. Gao et al. [48] have made significant strides in commonsense-based VQG, generating template-based visual questions that incorporate grounded and commonsense facts. The functional programming approach can create totally grounded questions using only the scene graph or compositional questions by merging the scene graph with external commonsense facts.

**D – Knowledge-Based**

This type of question also cannot be formed and answered using only the given image. It needs some external knowledge source for question generation and answering. Currently, few attempts have been made toward knowledge-based VQG. Knowledge-based VQA datasets like that of Shah et al. [49] has world-famous personalities and information about them. It has manual annotations for QA pairs. These questions can be answered using the image, question, and knowledge present in the form of a knowledge graph, e.g., Wikidata. Another knowledge-based dataset is the K-VQG [50]. In this dataset the model is given a knowledge triplet with a missing part and is expected to generate a question that can complement the missing part.

### 2.2.5 Summary

To summarize, in this section we analyzed various VQG models and categorized them across multiple dimensions: Question Generation Method, Learning Method, Input, and Type of Generated Questions. This categorization resulted in the following tables.

| | | | | | |
|---|---|---|---|---|---|
| Image + Context | CRIC [48], GQA [9] | COCO QA [41] | Multimodal differential network [11] | K-VQG [50], VAE approach [12] | |
| Image + Answer | | | iQAN [10] | | |
| Only Image | | | GRNN [47] | CLIP approach [13] | Discriminators approach [14] |
| | Template Based | Rule Based | Deep Neural Networks | Generative Networks | Reinforcement Learning |
| | | | Novel Question Generation (Automatic) | | |

**Table 2.1:** Table organizing all models and approaches according to the following categories: Question Generation Method, Learning Method and Input.

The table 2.1 reveals that models can be divided into three main categories. The traditional approaches, including template-based and rule-based approaches, do not use a learning method and were employed to generate the CRIC [48], GQA [9], and COCO QA [41] datasets. And the automatic approach that generates novel questions and can be further subdivided based on the employed learning methods, such as deep neural networks, generative networks, and reinforcement learning. An exception to this categorization is the visual curiosity model [8], which combines templates and reinforcement learning to generate its questions.

On the y-axis, we have the input that the model uses to generate the questions. We observe that, among the analyzed models, only iQAN [10] uses the answer as input to generate questions, while the other models use either the image alone or the image along with some context.

| | | | | | |
|---|---|---|---|---|---|
| Knowledge Based | | | | K-VQG [50], CLIP approach [13] | |
| Commonsense Based | CRIC [48] | | | VAE approach [12] | |
| Natural | | | Multimodal differential network [11], GRNN [47] | | Discriminators approach [14] |
| Totally grounded | GQA [9] | COCO QA [41] | iQAN [10] | | |
| | Template Based | Rule Based | Deep Neural Networks | Generative Networks | Reinforcement Learning |
| | | | Novel Question Generation (Automatic) | | |

**Table 2.2:** Table organizing all models and approaches according to the following categories: Question Generation Method, Learning Method and Type of Generated Questions.

This table 2.2 is similar to the previous one; however, instead of having the input on the y-axis, we have the type of generated questions. We observe a good mix of models for all types of questions, with no apparent correlation between the approaches and the types of questions generated. An interesting observation from both tables is that some models producing knowledge-based questions do not

require context as input. This is because these models were pre-trained with the necessary knowledge, eliminating the need for context at inference time. An example of this is the CLIP approach [13].

Overall, we analyzed a diverse collection of generation approaches across different categories, allowing us to develop a comprehensive view of the current state of the art in visual question generation and to inform the development of our own processes to tackle this task.

In tackling the task of sentence generation, our approach involved leveraging pre-existing models trained on similar tasks to capitalize on transfer learning 5.1.1. This decision was driven by the recognition that building a solution from the ground up would demand extensive computational resources and data, both of which we have in limited supply. In the subsequent subsection, we will delve into the models and datasets used during the development of our solution, elucidating their roles and contributions.

## 2.3 Models used on the training process

Throughout the solution's development, we employed two image-to-text models (GiT and Pix2Struct), as well as two text-to-text open generation models (Mistral-7B-Instruct-v0.2 and GPT-3.5 Turbo). Additionally, we indirectly used a dataset (Textcaps) by leveraging versions of GiT and Pix2Struct that were fine-tuned on this specific dataset.

### 2.3.1 GiT model

The Generative Image-to-text Transformer, GIT [15], represents a transformer decoder conditioned on both CLIP image tokens and text tokens. This model undergoes training using the "teacher forcing" technique on a plethora of (image, text) pairs. The primary objective of the model revolves around predicting the subsequent text token, given the image tokens and preceding text tokens.

The architecture, depicted in Figure 2.16, incorporates one image encoder and one text decoder. The image encoder is pre-trained with contrastive tasks (tasks that aim to learn representations by distinguishing between similar and dissimilar pairs of data.), employing an approach equivalent to sequentially addressing two tasks: firstly, using the contrastive task to pre-train the image encoder, followed by the generation task to pre-train both the image encoder and text decoder.

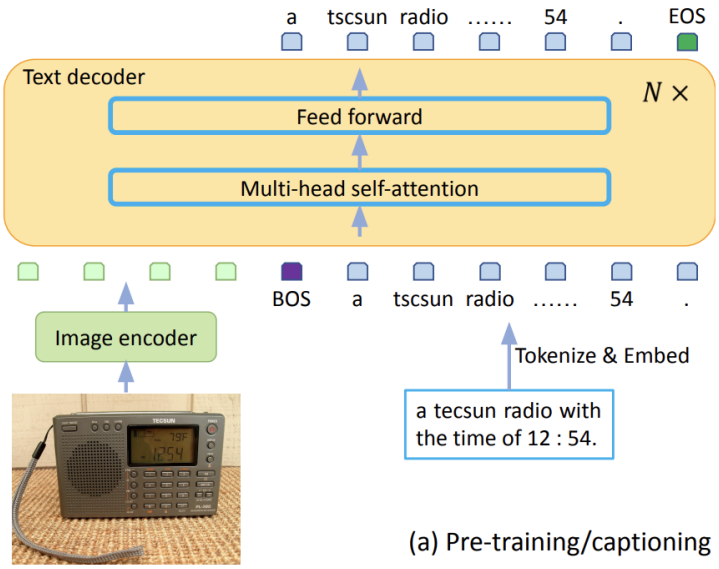



**Figure 2.16:** Network architecture of GIT, composed of one image encoder and one text decoder. From Wang et al. [15].

The text decoder is a transformer module to predict the text caption. The transformer module consists of multiple transformer blocks, each of which is composed of one self-attention layer and one feed-forward layer. Initially, the text is tokenized and embedded into D dimensions, followed by the addition of positional encoding and a layer normalization step. Subsequently, the image features are concatenated with the text embeddings to form the input for the transformer module. The decoding process begins with the [BOS] (Beginning of Sequence) token and is decoded in an auto-regressive way until the [EOS] (End of Sequence) token or reaching the maximum steps.

In terms of attention mechanisms, the model benefits from full access to the image patch tokens (”pieces” of an image, similar to tokens in text processing, that allow transformers to handle image data in a comparable manner to text data), facilitated by a bidirectional attention mask. This contrasts with a unidirectional attention mask, where not every image token can rely on all others. However, the model only possesses access to previous text tokens, employing a causal attention mask when predicting the subsequent text token.

We chose to integrate this model into our solution’s development because it achieves state-of-the-art results in image captioning, a task similar to ours, enabling us to leverage transfer learning effectively.

### 2.3.2 Pix2Struct model

The Pix2Struct model, as introduced by Lee et al. [17], stands as a pre-trained image-to-text model specifically tailored for visual language understanding, meaning that is capable of performing optical character recognition (OCR). It offers the flexibility of fine-tuning on tasks incorporating visually-situated

language. Pix2Struct is pretrained by learning to parse masked screenshots of web pages into simplified HTML.

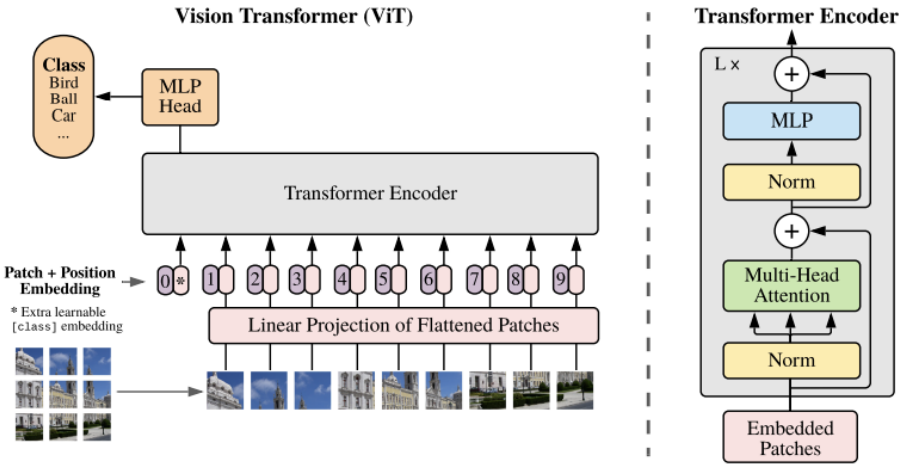



**Figure 2.17:** ViT overview. The model splits an image into fixed-size patches, linearly embeds each of them, adds position embeddings, and feeds the resulting sequence of vectors to a standard Transformer encoder. In order to perform classification, it uses the standard approach of adding an extra learnable "classification token" to the sequence. From Dosovitskiy et al. [16].

Built upon the foundations of ViT [16], Pix2Struct adopts an image-encoder-text-decoder architecture, illustrated in Figure 2.17. While the core architecture remains largely conventional, the authors introduce a minor yet significant alteration to the input representation. This tweak aims to enhance Pix2Struct's robustness to various forms of visually-situated language.

Before extracting fixed-size patches, the standard ViT scales the input images to a predetermined resolution, resulting in two undesirable outcomes: (i) distortion of the true aspect ratio, which can vary significantly, and (ii) complications when transferring these models to downstream tasks with different resolutions, as the model is trained solely on one specific resolution.

The authors instead suggest consistently scaling the input image either up or down to accommodate the maximum number of fixed-size patches within the designated sequence length. To ensure unambiguous handling of variables resolutions, 2-dimensional absolute positional embeddings are used for the input patches. These changes to the conventional ViT input mechanism yield two significant advantages: enhanced robustness against extreme aspect ratios and adaptability to on-the-fly changes in sequence length and resolution.

For Pix2Struct, the goal of pretraining is to represent the underlying structure of the input image. To that end, the authors create self-supervised pairs of input images and target text from web pages. For each page in the pretraining corpus, they start by collecting its HTML source and a screenshot using a viewport of 1024 x 1024, see figure 2.18.

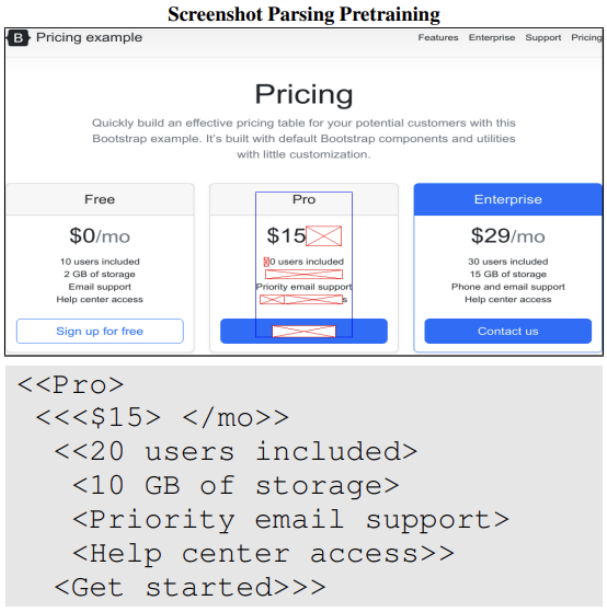



**Figure 2.18:** Example of the pretraining task (screenshot parsing). From Lee et al. [17].

Rather than directly training the model in the task of parsing screenshots, the authors opt for an initial ”warmup” stage where the model learns to read. This approach yields a robust curriculum learning effect, leading to greater stability and faster convergence during pretraining [17], along with improved performance during fine-tuning. They accomplish this by presenting the model with images of text snippets rendered in random colors and fonts, training it to decode the original text.

We chose to integrate this model into our solution's development for reasons similar to those driving the adoption of the GiT model. Additionally, the model's unique training strategy of initially focusing on reading before delving into representation of image structure proved instrumental. This approach yielded exceptional results, particularly in the task of recognizing variables within charts, as we will explore in subsequent sections.

### 2.3.3 Textcaps

TextCaps [18] stands as a groundbreaking dataset designed for image captioning augmented with reading comprehension. Unlike traditional image captioning tasks, TextCaps requires models to not only generate captions but also to comprehend and reason over text embedded within the images. This unique challenge requires models to integrate a novel modality of textual information present in images while simultaneously reasoning over visual content to generate accurate image captions. The dataset proves invaluable for training models to recognize and understand text within scenes, as many existing state-of-the-art image captioning methods primarily focus on visual objects, often neglecting textual elements. A selection of images alongside their corresponding captions from this dataset is illustrated in

Figure 2.19.


**Figure 2.19:** Illustration of TextCaps captions. From Sidorov et al. [18].

Using versions of the GiT and Pix2Struct models fine-tuned on this dataset allows for more effective use of transfer learning. This is particularly advantageous in facilitating our task of identifying variables in data charts, which bear resemblance to challenges posed by TextCaps. By leveraging pre-trained models adapted to this dataset, we can enhance the performance of our system in recognizing and reasoning about textual elements within images, thus improving overall results.

### 2.3.4 GPT-3.5 Turbo

Developed by OpenAI, GPT models [51] leverage transformer architecture to process sequential data, particularly suited for tasks involving language understanding and generation. The introduction of GPT marked a significant departure from traditional language models, offering unparalleled performance across various NLP benchmarks.

At the core of GPT lies a multi-layer transformer decoder architecture, comprising self-attention mechanisms and feed-forward neural networks. This architecture enables the model to effectively capture long-range dependencies within input sequences, facilitating robust language understanding and generation. Notably, GPT employs a left-to-right autoregressive decoding strategy, where each token in the output sequence is generated conditioned on preceding tokens. This mechanism allows for the

generation of fluent and contextually coherent text across diverse contexts and tasks.

The training regime of GPT involves pre-training on large-scale text corpora, followed by fine-tuning on specific downstream tasks. During pre-training, the model learns to predict the next token in a sequence given preceding tokens being a autoregressive model [52]. Fine-tuning further adapts the learned representations to target tasks, resulting in performance gains and task-specific optimization.

GPT models have demonstrated exceptional performance in tasks such as text generation, language translation, sentiment analysis, and question answering. GPT-3.5 Turbo in particular, at the start date of this work, is the most recent version of these models that we are able to finetune without requirements since GPT-4 fine-tuning is in experimental access for only eligible developers. Integrating this model into our solution for the task of generating true or false sentences affords us the opportunity to fine-tune it with less data, thereby enhancing its creative capabilities in sentence generation. Leveraging transfer learning, the model draws upon its training, allowing it to generate sentences imbued with understanding of the domain, thus improving the overall performance of our solution.

### 2.3.5 Mistral 7B

Mistral 7B, [19] is a 7–billion-parameter language model that outperforms the 13B parameters models like Llama 2 across all evaluated benchmarks and outperforms 30B parameter models like Llama 1 in reasoning, mathematics and code generation.

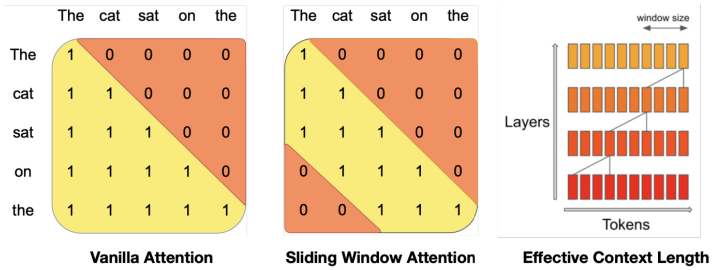



**Figure 2.20:** Sliding Window Attention. The number of operations in vanilla attention is quadratic in the sequence length, and the memory increases linearly with the number of tokens. To alleviate this issue, the authors use sliding window attention. Note that tokens outside the sliding window still influence next word prediction. At each attention layer, information can move forward by W tokens. Hence, after k attention layers, information can move forward by up to k $\times$ W tokens. From Jiang et al. [19].

To obtain this benchmarks results the authors introduce three changes:

- Sliding Window Attention, see figure 2.20: SWA allows a transformer model to consider information beyond a fixed window size. At each layer, hidden states can access information from previous layers within a range extending beyond the window size. This technique enables the model to effectively capture dependencies across long sequences. For instance, with a window size of 4096, the model theoretically has an attention span of approximately 131K tokens. In practice,

using SWA with specific adjustments results in a 2x speed improvement compared to a standard attention mechanism.

- Rolling Buffer Cache: To manage memory usage efficiently, a rolling buffer cache is introduced. This cache has a fixed size matching the window size (W). Keys and values for each timestep are stored in positions determined by the modulo operation of the timestep index. As new information is processed, older values in the cache are overwritten, preventing the cache size from growing indefinitely. For example, on a sequence length of 32k tokens, implementing a rolling buffer cache with a window size of 3 reduces cache memory usage by 8x without compromising model quality.

- Pre-fill and Chunking, see figure 2.21: Before generating a sequence, the model can pre-fill the cache (a storage area for key-value pairs) with the prompt, which is the input provided to the model in advance. This way, the model has access to the prompt information from the beginning of the sequence generation process. If the prompt is very large, it can be divided into smaller pieces or ”chunks”. Each chunk can then be pre-filled into the cache separately. The size of each chunk can be determined by the window size. The model then computes attention over both the cache and the chunk, allowing it to efficiently process and generate sequences.

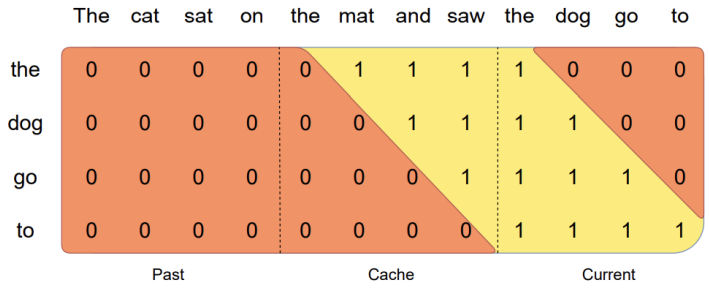



**Figure 2.21:** Pre-fill and chunking. During pre-fill of the cache, long sequences are chunked to limit memory usage. The authors process a sequence in three chunks, “The cat sat on”, “the mat and saw”, “the dog go to”. The figure shows what happens for the third chunk (“the dog go to”): it attends itself using a causal mask (rightmost block), attends the cache using a sliding window (center block), and does not attend to past tokens as they are outside of the sliding window (left block). From Jiang et al. [19].

We opted for this model in our solution due to its smaller size (7 billion parameters compared to GPT-3.5’s 175 billion), yet comparable performance to models with 30 billion parameters on certain benchmarks. This choice offers a valuable contrast to our results when compared with a larger model like GPT-3.5, as larger isn’t always better for specific tasks, such as generating true or false sentences. Should this smaller model outperform GPT-3.5, we can apply Occam’s razor principle and select it. Unfortunately, as we will detail in a subsequent section, this model did not yield satisfactory performance for the task at hand.

# 3

# Problem Statement

In recent years, there has been a notable and transformative trend towards the massification of education. Education massification represents a significant paradigm shift in the field of education, characterized by the widespread accessibility of educational opportunities and resources, largely facilitated by advancements in technology. This trend, primarily driven by the proliferation of online learning platforms and open educational resources, possesses the potential to revolutionize traditional educational practices by fostering inclusivity, diversity, and flexibility in learning.

Nevertheless, it is essential to acknowledge that education massification brings with it a set of challenges that require careful consideration and resolution. As a consequence of this phenomenon, class sizes have expanded to unprecedented numbers, often unattainable in a traditional face-to-face teaching environment. This, in turn, diminishes the level of individual interaction between students and educators, thereby potentially relegating the student to a passive role, which can lead to a loss of interest and require greater efforts on the part of teachers to maintain student engagement.

Moreover, the abundance of online courses and the wide array of available materials and activities can overwhelm students [20], making it difficult for them to make informed choices regarding their learning paths. The need for personalization in learning resources has thus emerged as a critical requirement to ensure equitable educational opportunities for all and address these issues effectively. Personalized learning pathways, tailored to individual needs and preferences, hold the potential to bridge educational

disparities, enabling learners to progress at their own pace [21] and in accordance with their unique learning styles. Personalization, therefore, represents a pivotal step toward creating a more equitable and inclusive educational landscape. One notable manifestation of the massification of education is the increased utilization of Massive Open Online Courses (MOOCs), which are being adopted by universities and corporate entities alike [22]. MOOCs serve as a means to disseminate course-related materials in an appropriate online space, accessible to the target audience, such as university students enrolled in a particular course.

The rise of these MOOC platforms coincided with the general expansion of online learning. Blended learning, an approach that combines traditional face-to-face instruction with digital education, has been very popular particularly within universities [23]. Moreover, by 2013, approximately 70% of educational institutions in the United States attested to the fundamental role of online learning in their long-term strategies [53]. The COVID-19 pandemic further increased a shift towards full-scale online education [54].

Over the years, the development of MOOCs has faced various challenges, whether in their inception or ongoing operation. Among these challenges, personalization has emerged as a heavily researched domain [24], aiming to create a learning experience as unique as possible for each student. However, personalization hinges on the availability of a diverse array of learning resources, particularly challenging to generate manually, especially evaluation items like textual questions or intricate exercises. As observed by Kurdi et al. [25], the need for a ”continuous supply” of these resources has birthed a nascent research domain in automatic question generation.

Consequently, there exists a pressing demand for the automated generation of learning resources and exercises tailored to the unique needs of each user. In particular, the generation of personalized questions with specific characteristics offers numerous benefits, including the presentation of a virtually endless array of slightly diverse versions of the same problems to students. This approach has two clear use cases: firstly, it enables students to confront problems relevant to areas in which they may not excel, fostering a deeper understanding of those subjects. Secondly, it allows for the creation of individualized assessment materials, ensuring that assessment tasks remain distinct for each student.

Data charts in particular serve as indispensable tools in data visualization, widely employed in data science, statistics, and machine learning courses. By engaging with these visual representations, students cultivate a profound understanding of the data they scrutinize. For instance, a boxplot effectively portrays the distribution of data, highlighting skewness, spread, and the occurrence of outliers within the dataset.

In the realm of teaching, these data charts are often accompanied by questions that delve into the underlying dataset. However, manually crafting these questions proves time-consuming and hinders the provision of personalized questions tailored to individual students.

Our goal is to leverage recent advances in natural language processing and computer vision to propose a methodology for training models that can generate true/false sentences given a data chart and classify a given sentence as true or false based on the chart.

To reach this objective we divided our approach in three phases:

- The first phase involves generating the data that will be used to train the model.

- The second phase focuses on training the model to generate questions using that data.

- The third phase entails training a model to answer the questions given an image-question pair.

In Figure 3.1, the entire process is illustrated, showing how each task interacts with the others. The first task creates the dataset, which is subsequently used by two other tasks to train both the question generation and question answering models.

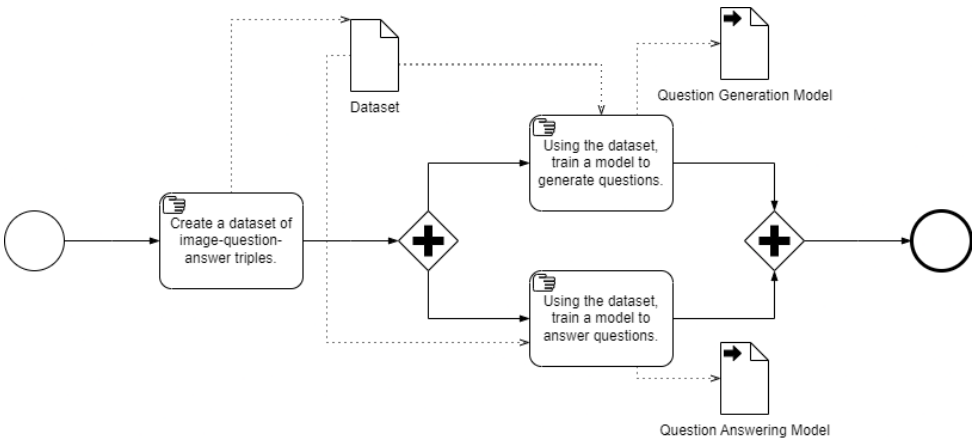



**Figure 3.1:** BPMN diagram showing the process of this work and the inputs and outputs of each task.

Figure 3.2 demonstrates the input and output of each model during inference time. The question generation model takes an image as input and outputs a question, while the question answering model takes both an image and a question as input and outputs an answer.

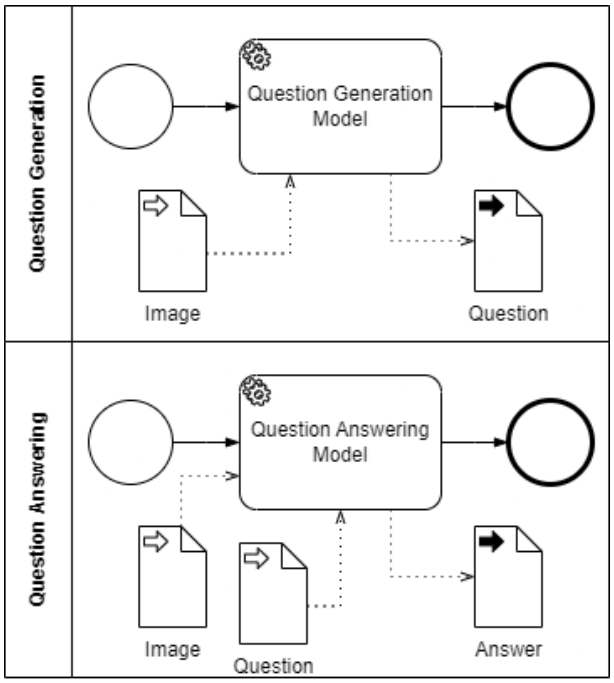



**Figure 3.2:** BPMN diagram showing each model at inference time.

In the following chapters, we will address each one of these phases in detail, describing the approach used and evaluating their results.

# 4

# Data Generation

## Contents



The first phase requires a dataset consisting of Image-Question-Answer triples. This is necessary because we want the question generation model to produce questions that are non-trivial and require knowledge about data science. Therefore, we need a dataset with questions of this nature to train the model effectively. If we only aimed to develop a question generation model, a dataset of Image-Question pairs would suffice. However, since we also aim to develop a question answering model, our dataset must include the answers as well. This allows us to train the question answering model to respond accurately to these data science-related questions.

Since, to the best of our knowledge, there is currently no existing dataset of data science sentences composed of triples (Image-Question-Answer), we must generate it.

In the next sections, we will detail the main dataset, which consists of Image-Question-Answer triples, as well as the auxiliary datasets used to train specific components of our solutions, derived from the main

dataset.

## 4.1 Data supporting the questions

Before delving into the creation of the questions, let's examine the data that supports them. We used a template-based approach to generate the dataset, filling in templates with various combinations of details (e.g., class names, variable names, number of records) from different datasets. Therefore, our first task is to create these templates.

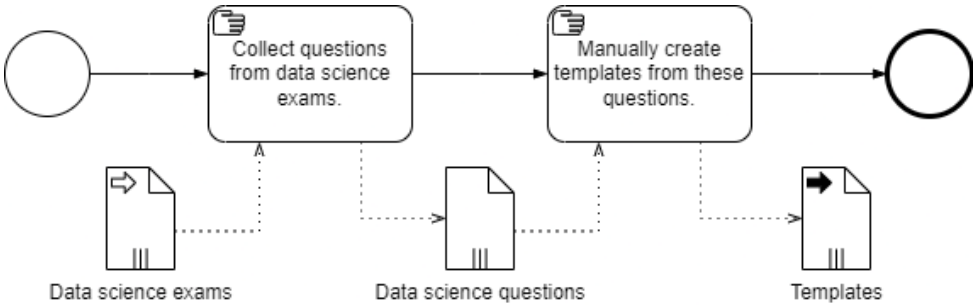



**Figure 4.1:** Process showing the creation of the templates.

In Figure 4.1, we illustrate the process of creating these templates. The process begins by collecting questions from data science course exams at Instituto Superior Técnico. Each question is then manually analyzed to create templates. For example, consider the following exam question: "The accuracy for the presented tree is higher than its recall.". From this question, we created the template: "The [1] for the presented tree is [2] than its [3].".

| Template | Space1 | Space2 | Space3 |
|---|---|---|---|
| The [1] for the presented tree is [2] than its [3]. | [accuracy,recall, precision,specificity] | [higher,lower] | [accuracy,recall, precision,specificity] |
| The variable [1] can be coded as ordinal without losing information. | [<all_variables>] | | |
| Variable [1] presents some outliers. | [<numeric_variables>] | | |

**Table 4.1:** Three examples from the collection of templates.

In table 4.1, we present three examples from our collection of templates. Each number in brackets represents a placeholder, and along with the template itself, we include additional columns within the same record that list possible values to fill these placeholders.

In the first example, all possible values are hardcoded. In the second example, instead of hardcoded variables, there is an indication that all variable names from a given dataset can be used to fill this space. In the third example, only numeric variables from a given dataset can be used to fill the space. These

special indications are enclosed in "<>" to differentiate them from hardcoded values. These indications will be used to generate the questions, as will be explained in Section 4.3.

This process resulted in the creation of 74 templates.

After developing the templates, we needed datasets to populate them. We collected 34 datasets from the UCI ML repository and Kaggle, ensuring a mix of numeric, binary, symbolic, and ordinal variables to allow for a diverse range of possible questions.

From this process we obtained a set of templates and a collection of datasets. In section 4.3 we will explain with detail how we used these two artifacts to create the main dataset. In the next section we will explain the data charts needed for creation of the dataset.

## 4.2 Plot generation

To generate the data charts used to create the dataset, we used the 34 datasets we collected. We used Python [55], Pandas [56] and Matplotlib [57] to generate for each dataset the following charts:

- Bar chart displaying the number of records and the number of variables.
- A correlation heatmap showing the correlation between all numeric variables.
- A set of histograms for the numeric variables, aggregated into a single image.
- A set of bar charts for the symbolic and binary variables, aggregated into a single image.
- A set of boxplots for the numeric variables. In one image we aggregate all boxplots representing each numeric variable.
- Bar chart illustrating the distribution of missing values for each variable.
- Bar chart showing the explained variance ratio for each principal component.
- A decision tree diagram with a depth of 2, where both nodes at depth 1 share the same condition. For example, if the condition at depth 0 is "A>20" (true or false), the condition at depth 1 would be "B<0" for either case. This restriction allows us to formulate questions about Naive Bayes and KNN classification.
- Bar chart representing the class variable.
- Six multi-line charts depicting the accuracy for several models trained on these classification datasets. These models include:
  - Chart showing the test and train accuracy of decision trees according to the depth of the tree.

– Chart showing the test and train accuracy of k-nearest neighbors (KNN) according to the number of neighbors (K).
– Chart showing the test and train accuracy of random forests according to the number of estimators.
– Chart showing the test and train accuracy of gradient boosting according to the number of estimators.
– Chart showing the test and train accuracy of multi-layer perceptrons according to the number of iterations.
– Chart showing the test and train accuracy and recall of decision trees according to the depth of the tree. This one is similar to the first chart however it also has the train and test recall allowing us to make questions about the relationship between recall and accuracy.

This process yielded 459 plots, with 15 charts generated for each dataset. Exceptions were made for datasets without missing values or symbolic variables, where the corresponding bar charts were not generated.

At this stage, we have the templates, datasets, and plots ready. In the next section, we will explain how these three artifacts were used to create our main dataset.

## 4.3 Main Dataset

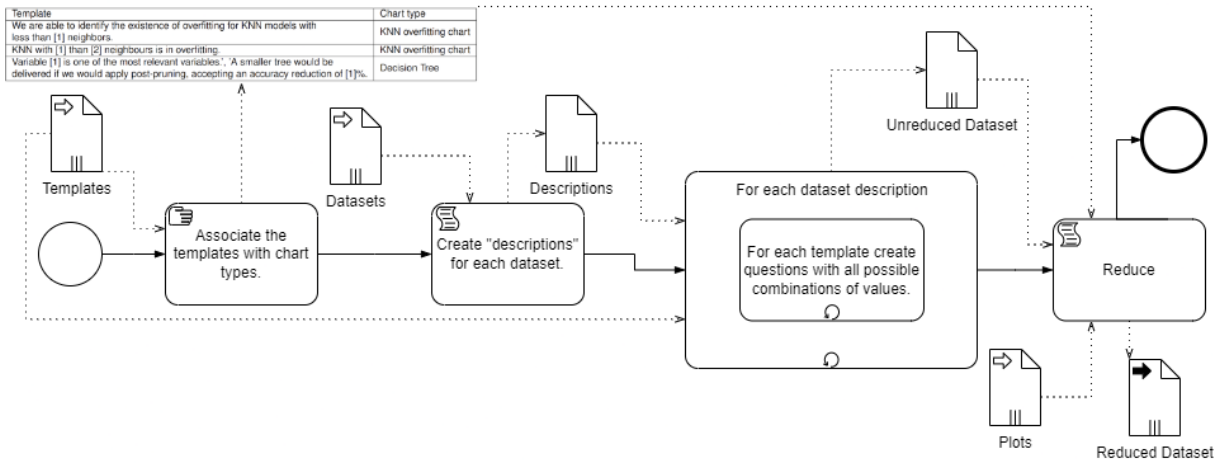



**Figure 4.2:** Dataset creation process.

As mentioned earlier, our goal is to create a dataset composed of Image-Question-Answer triples. In Figure 4.2, we outline the complete dataset creation process. We started with 459 different charts of 15 different types and 74 templates. The first step was to associate each template with its corresponding

chart type. For example, the template ”The [1] for the presented tree is [2] than its [3].” requires a decision tree to be answered, so we associate this template with that chart type. This process of associating each template with a chart type was done manually for all 74 templates.

After associating each template with its corresponding chart type, we proceeded to fill the templates. For this, we created a ”description” for each dataset containing all the necessary information to fill in the placeholders in the templates according to the corresponding indications (as explained in Section 4.1).

| | |
|---|---|
| Number of Records | 10001 |
| Number of variables | 11 |
| Target Variable | Exited |
| Target values | [0,1] |
| Numeric Variables | [CreditScore, Age, Tenure, Balance, NumOfProducts, EstimatedSalary] |
| Binary Variables | [Gender, HasCrCard, IsActiveMember] |
| Symbolic Variables | [Geography] |

**Table 4.2:** Example of a description for one of the collected datasets.

In Table 4.2, we see that each description includes the number of records, the number of variables, the target variable and its values, a list of numeric variables, a list of binary variables, and a list of symbolic variables. We used the Pandas library [56] to manipulate each dataset and obtain this data.

To create the questions, we used a nested loop: the outer loop iterated through each dataset description, and the inner loop iterated through each template. For each template, we created questions by filling in the placeholders with all possible combinations of values (these possible values were defined during the template creation process in Section 4.1), with some exceptions to avoid repeated variable names in the same question. Since we knew the chart type required by each template, we associated the generated questions with the correct plot from the current dataset, obtaining Image-Question pairs.

After this process, we found ourselves with a dataset containing 130,352 of Image-Question pairs. However, we opted to reduce the dataset for several reasons:

- Firstly, certain templates entail more spaces to fill, resulting in a higher number of value combinations and consequently more generated sentences for that template. This poses a risk of overfitting to a particular template.
- Secondly, certain variable names are more prevalent than others. For instance, the variable ”age” appeared frequently across the datasets we used. Consequently, this variable name is present in numerous generated sentences, potentially leading to overfitting as the model may learn to generate sentences primarily featuring this variable.
- Thirdly, managing a dataset of this magnitude would necessitate significant time and effort to manually label the sentences as true or false. This labeling process is crucial for training the visual question answering model, making dataset reduction imperative.

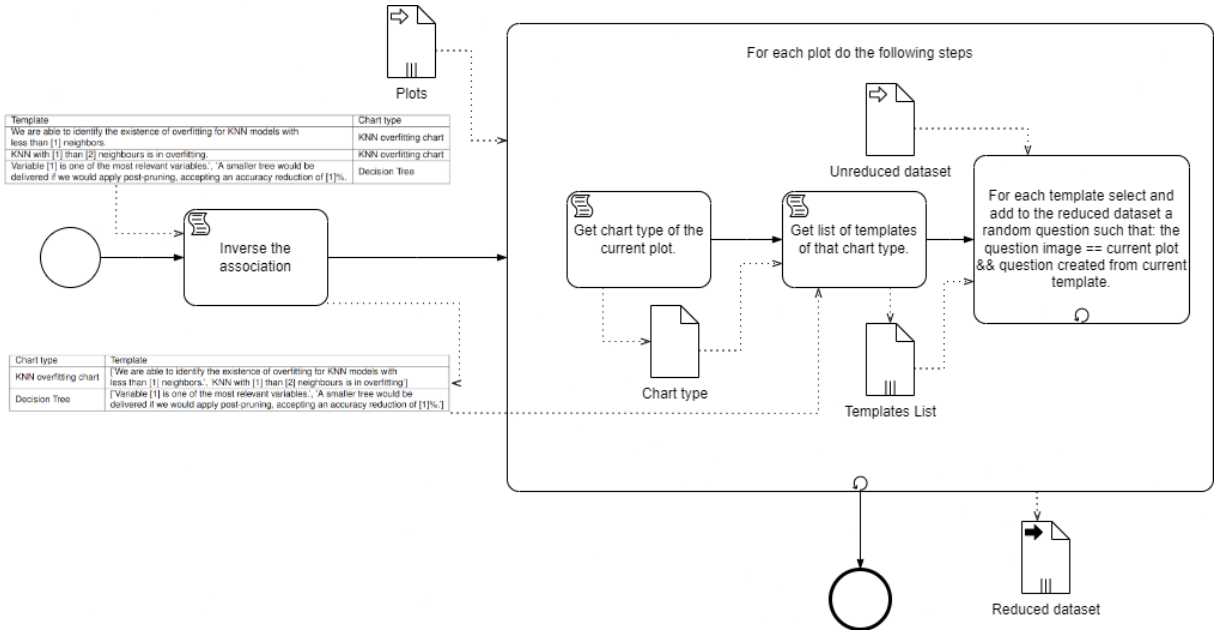



**Figure 4.3:** The reduce process in detail.

To address these concerns, we followed a systematic process. As illustrated in Figure 4.3, we first reversed the association, giving each chart type a list of templates. We then iterated through each of the 459 plots, performing the following steps:

- Identify the chart type using the plot name.
- Select the corresponding list of templates for that chart type.
- Iterate through that list of templates and, for each template, randomly select a question from the unreduced dataset that meets the following criteria: it uses the current template and was generated for the current plot.

After this process, we ended up with a reduced dataset of 2,630 Image-Question pairs, ensuring a balanced representation of templates.

The last step to complete the dataset was to answer each one of Image-Question pairs. This process was not automatic, we manually annotated each question to complete the dataset resulting on a dataset of 2,630 Image-Question-Answer triples.

### 4.3.1 Dataset Statistics

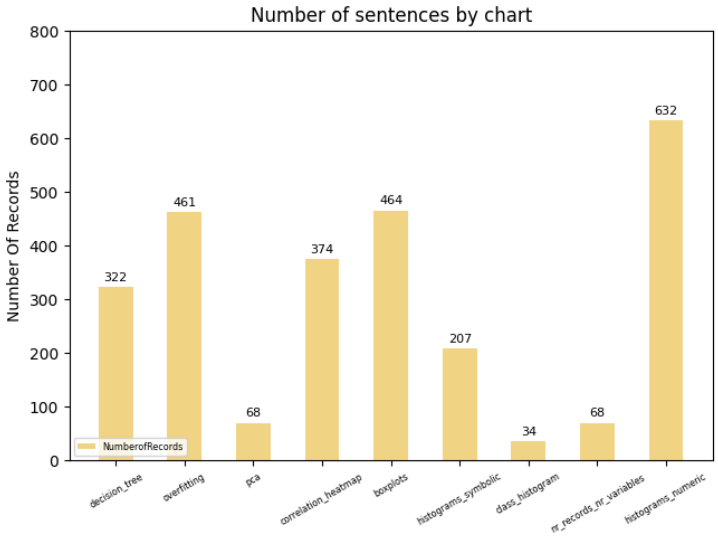



**Figure 4.4:** The number of sentences by chart.

The dataset does not have an equal number of sentences for each chart type, as shown in Figure 4.4. This discrepancy arises because some charts allow for a more diverse range of questions. For instance, with the class bar chart, we can only ask if the dataset is balanced, whereas the boxplot enables a broader variety of questions, leading to a larger number of sentences.

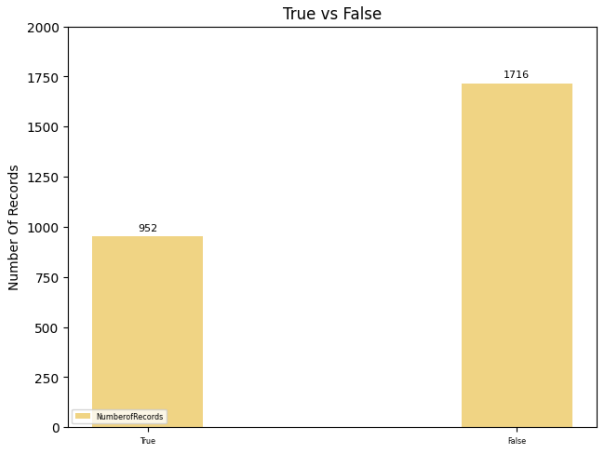



**Figure 4.5:** The distribution of sentences by true and false.

The dataset is unbalanced, containing more false sentences than true ones, as illustrated in Figure 4.5. However, this imbalance did not pose a problem during training. We continuously monitored the metrics to ensure that the model did not exhibit a bias towards false classifications.

## 4.4 Auxiliary datasets

Throughout the development process, we created several auxiliary datasets derived from the main dataset. In this section, we will provide an overview of each of these datasets. For illustrative examples of records from both the main dataset and the auxiliary datasets, please refer to the appendix A. Let's briefly explain each dataset, their purposes, and the structure of their records.

- **Chart Caption Dataset**: During the development of our question generation model, we felt the need to divide the task into two subtasks: generating a caption or description of the chart, and then generating a question based on that caption. This need originated the creation of this dataset to train a model for the first subtask. Each entry in this dataset consists of an image of a chart and its corresponding caption. For details on how this dataset was constructed, refer to subsection 4.4.1.

- **Caption Templates Dataset**: After analyzing the results of the training with the Chart Caption Dataset, we identified an opportunity for further improvement by dividing the caption generation task into two subtasks. The first subtask involves classifying the type of chart (e.g., boxplot, histogram) and assigning it a general caption template that does not include the variable names from the chart. The second subtask focuses solely on identifying the variable names. This dataset originates from the first subtask, enabling us to train a model to classify the input chart and assign an appropriate general template. Each record in this dataset comprises an image of a chart and its corresponding general template. For details on the construction of this dataset, refer to subsection 4.4.2.

- **Chart Variables Dataset**: This dataset is derived from the second subtask described previously, where the goal is to train a model to identify the variable names in the input chart, allowing us to replace placeholders in the general template with these variable names. Each record in this dataset includes an image of a chart and a list of the variables represented within that chart. For details on the construction of this dataset, refer to subsection 4.4.3.

- **Chart Data Dataset**: While the previous auxiliary datasets were created to develop the question generation model, this dataset was developed to aid in the creation of the question answering model. During the training of the question answering model, we realized the necessity of extracting more information than just the variable names and the type of chart to answer the questions effectively. Therefore, this dataset was created to train a model to extract the required information from the input chart. Each entry in this dataset consists of an image of a chart and its corresponding data. The specifics of the data that need to be extracted vary depending on the type of chart. For details on the data extraction specifics, refer to subsection 4.4.4.

### 4.4.1 Chart Caption Dataset

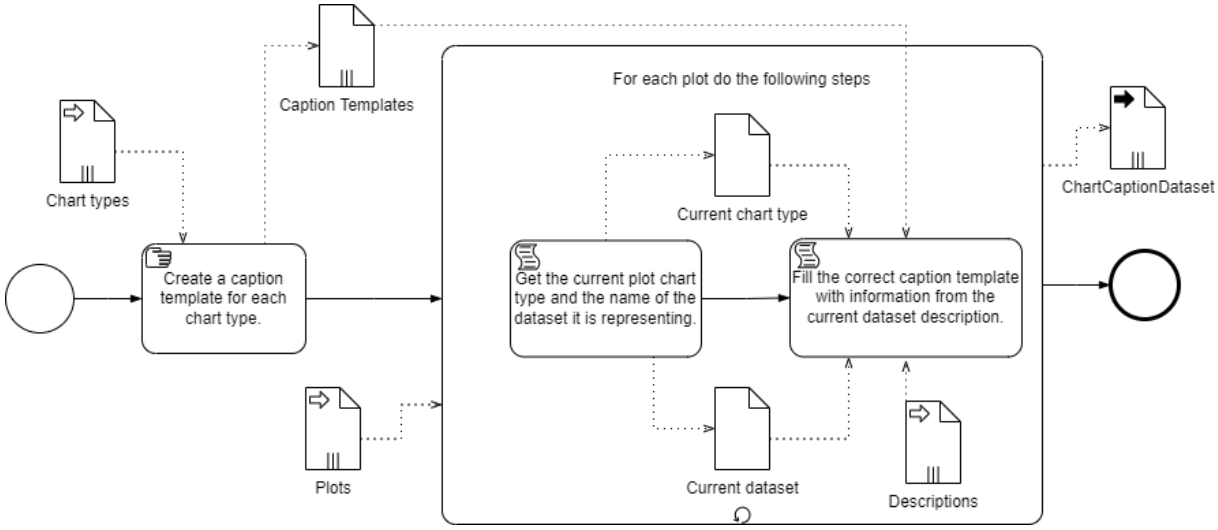



**Figure 4.6:** The process of construction of the Chart Caption Dataset.

The process of constructing this dataset is illustrated in Figure 4.6. We began by creating a caption template for each of the 15 chart types. For example, the caption template for a set of boxplots is ”A set of boxplots of the variables [].” Each caption template encapsulates two important elements: the chart type name (”A set of boxplots” in this case) and a placeholder (identified by square brackets) for the information needed to generate questions about the plot.

This dataset is designed to train a model to generate captions for charts, which will subsequently be used to generate questions. Therefore, both the chart type name and the information about the chart are essential. The information itself is a list of the variables represented in the plot, except for the bar chart of explained variance ratio, where the information is the number of principal components in the plot. For the overfitting multi-line charts and the bar chart displaying the number of records and the number of variables, we do not use placeholders, as we only require the chart type itself to generate the questions, and the chart type is already included in the caption template.

After creating these caption templates, we iterated through each of the 459 plots and performed the following steps:

- First we extracted the chart type and the name of the current dataset from the chart, this was possible because the names of all charts follow the convention ”<dataset name>_<chart type>”, for example ”diabetes_boxplots”.

- Then, knowing the dataset name, we retrieved its description and replaced the placeholder with information from the description, thus creating the caption for the current plot.

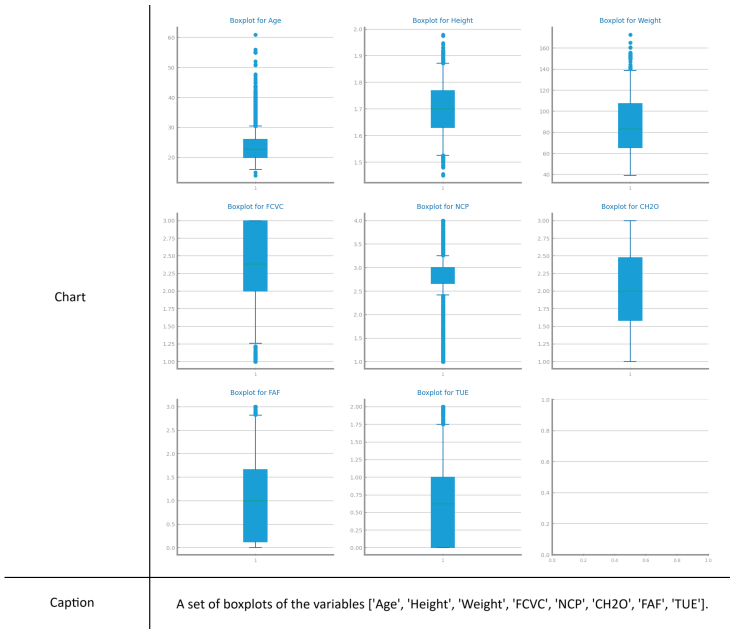



**Figure 4.7:** A record from the Chart Caption Dataset.

The Chart Caption dataset comprises Image-Caption pairs, where each image is accompanied by its corresponding caption. This dataset consists of 459 records, matching the number of charts we generated. For an example, refer to the record in Figure 4.7. For additional records from this dataset, refer to Appendix A, where you can see all 15 caption templates and the information that fills the placeholders for each chart type.

### 4.4.2 Caption Templates Dataset

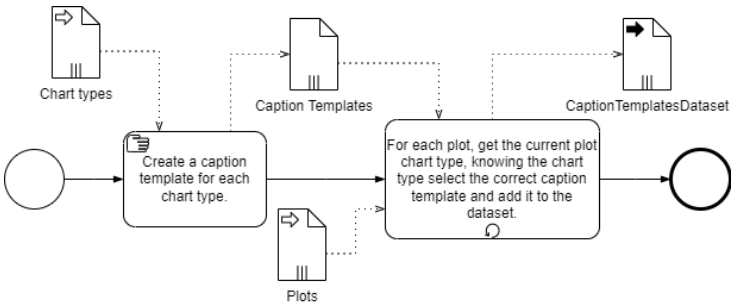



**Figure 4.8:** The process of construction of the Caption Templates Dataset.

The need for this dataset emerged after dividing the caption generation task into two parts, as explained earlier. Its construction is similar to the Chart Caption Dataset, with the key difference being that we do not replace the placeholders with information from the dataset description, as shown in the construction process in Figure 4.8.

This effectively makes it a classification dataset since we only have 15 caption templates (one for each chart type). The purpose of the model trained with this dataset is to classify the chart by attributing to it the proper caption template.

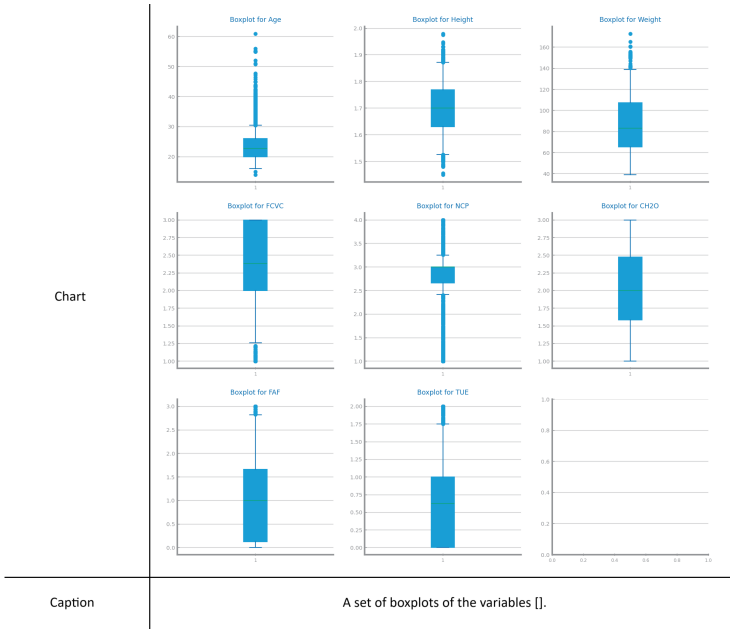



**Figure 4.9:** A record from the Chart Templates Dataset.

Similar to the Chart Caption Dataset, each record contains an image and its caption. However, in this dataset, the captions do not include the variables represented in the chart; instead, placeholders are used. An example record is depicted in Figure 4.9.

### 4.4.3 Chart Variables Dataset

As explained earlier, this dataset results from the division of the caption generation task. The Caption Templates Dataset addresses the the task of classifying the plot and assigning the appropriate caption template, while this dataset serves to train a model to identify the information represented in the plot.

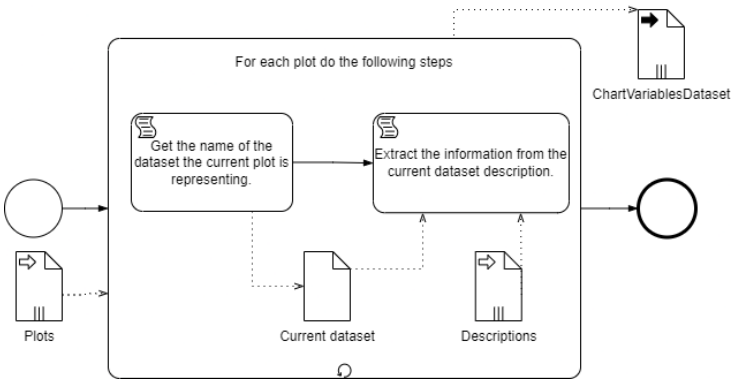



**Figure 4.10:** The process of construction of the Chart Variables Dataset.

The construction of this dataset is similar to the Chart Caption Dataset, with the main difference being that we do not need to identify the chart type to get the correct caption template. Instead, we only need the dataset name to extract information from the dataset description. As shown in Figure 4.10, for each plot, we get the name of the dataset it represents (which is straightforward due to the naming convention explained earlier) and use the dataset name to extract the information from the dataset description.

The information, as explained in the Chart Caption Dataset section 4.4.1, is a list of the variables represented in the plot, except for the bar chart of explained variance ratio, where the information is the number of principal components in the plot.

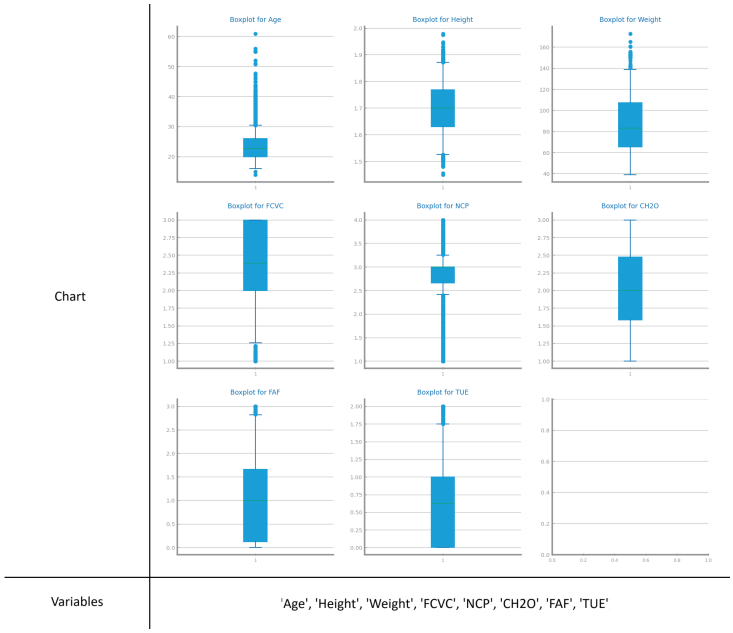



**Figure 4.11:** A record from the Chart Variables Dataset.

To summarize, the Chart Variables Dataset is dedicated to training a model to identify the variables in a chart. Each record in this dataset comprises a chart along with a list of variables represented within

that chart, except for the bar chart of explained variance ratio, where the information is the number of principal components in the chart and the primary objective of this dataset is to train a model to identify the variables depicted in data charts. An illustrative example of this dataset is provided in Figure 4.11. For additional records, refer to Appendix A.

This dataset and the Caption Templates Dataset work together to allow for training a model capable of generating a caption given a plot. They are essentially the result of separating the concerns of the Chart Caption Dataset.

### 4.4.4 Chart Data Dataset

This auxiliary dataset differs from the previous ones, as it was created for the visual question answering task rather than for visual question generation. The need for this dataset arose when we realized that answering questions about the charts required more information than just the variable names represented in the plot. Therefore, we created this dataset to capture all the information needed to answer questions for each chart type.

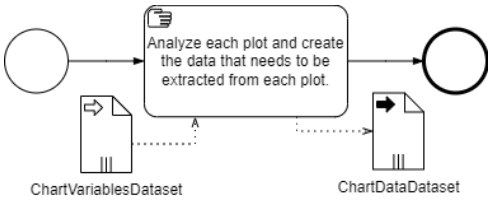



**Figure 4.12:** The process of construction of the Chart Data Dataset.

The creation process follows a manual approach. While the other auxiliary datasets were mostly created automatically (with only the 15 caption templates requiring manual work), this dataset required more manual effort to annotate. As shown in Figure 4.12, we started with the Chart Variables Dataset and manually analyzed each plot to extract the data that the model needs to identify.

The data extracted varies by chart type. Here is the data extracted for each chart type:

- **Number of Records vs Number of Variables**: We extract the number of records and the number of variables in the format: "[<nr_records>,<nr_variables>]".

- **Correlation Heatmap**: We extract the number of variable and the variable pairs that have a correlation score > = 0.80, in the format "{<number of variables>, [<variable pair 1>,<variable pair 2>,...]}".

- **Numeric Variables Histograms**: For each variable, we note whether it has outliers, is balanced, and is ordinal (this can be detected by the space between the bars, if they have no space between

them it is not ordinal if there is space between them it is ordinal). The format is: "{<variable name>: {Outliers:<T or F>, Balanced:<T or F>, Ordinal:<T or F>}, ...}".

- **Symbolic Variables Bar Charts**: We extract the variables and their possible values in the format: "{<variable name>: [<Value 1>,<Value 2>, ...], ...}".
- **Numeric Variables Boxplots**: We determine if the variables are normalized and whether each variable has outliers and is balanced. The format is: "[¡T or F¿, {<variable name>: {Outliers:<T or F>, Balanced: <T or F> }, ...}]".
- **Missing Values Bar Charts**: We extract the variable name and the corresponding number of missing values in the format: "{<variable name>: [<number of missing values>, ...}".
- **Bar Chart for Explained Variance Ratio**: We list the explained variance ratio for each principal component: "[<Explained Variance Ratio for Principal Component 1>, ...]".
- **Decision Tree**: We extract each node's condition, samples, value, class, true path, and false path in the format: {<Condition>:{samples: <Number of samples>, value:[<number of samples of first class value>, ...],class:<majority class in this node>,True:<node for true condition>,False:<node for false condition>}}. Refer to figure 4.13 to see an example of a decision tree record.
- **Bar Chart for Class variable**: We extract the number of records for each class in a list format: "[<number of records for first class>, ...]";
- **Multi-line Chart Decision Tree**: We extract the depth where overfitting starts (visually identified by a descending slope on the test accuracy line) in the format: "overfitting after <depth>". If no overfitting occurs, we put "no".
- **Multi-line Chart KNN**: We extract the number of neighbors where overfitting ends in the format: "overfitting until <number of neighbors>". If no overfitting occurs, we put "no".
- **Multi-line Chart Random Forests**: We extract the number of estimators where underfitting ends, the number of estimators where overfitting starts, and whether the test accuracy is greater, smaller, or equal to 0.80. The format is "underfitting until <number of estimators>,overfitting after <number of estimators>,acc<'<0.8' or '>0.8' or '=0.8'>". We extract more information of this chart because the questions of random forests we collected from the data science exams this information is needed to answer.
- **Multi-line Chart Gradient Boosting**: We extract the number of estimators where overfitting starts in the format: "overfitting after <number of estimators>". If the chart does not enter in overfitting we just put "no".

- **Multi-line Chart MLP**: For this chart we extract the number of iterations where overfiting starts to happen, so the format is: ”overfitting after <number of iterations>”. If no overfitting occurs, we put ”no”.
- **Multi-line Chart accuracy and recall for decision tree**: This chart is used only in trick questions regarding the relation between accuracy, recall and overfitting so we do not need extract the depth of overfitting to answer and only use the token ”acc_rec” to identify the chart. If we wanted to identify the exact depth of overfitting on this chart it would be harder for the trained model to do it since there are more lines in the chart in comparison to the simple overfitting decision tree chart.

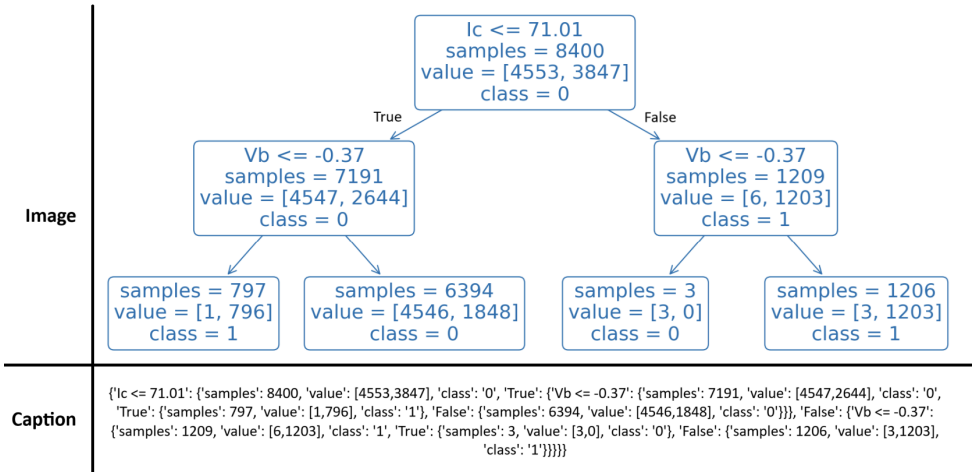



**Figure 4.13:** An example of a record in the Chart Data Dataset.

To summarize, the Chart Data Dataset is used to train a model to extract the necessary information from charts to classify sentences about them as true or false. It serves as an auxiliary dataset for the question answering task. This dataset was manually created using the same 459 plots used in the other auxiliary datasets, resulting in a dataset containing 459 records. Each record in the dataset consists of a chart and the information that needs to be extracted from it, as illustrated in Figure 4.13, for more records refer to the appendix A.

# 5

# Sentence Generation

## Contents



In tackling the task of sentence generation, our approach involved leveraging pre-existing models trained on similar tasks to capitalize on transfer learning 5.1.1. This decision was driven by the recognition that building a solution from the ground up would demand extensive computational resources and data, both of which we have in limited supply. In the subsequent section, we will delve the the into the training process of the question generator analyzing what composes each component of our solution.

## 5.1 Global training process

During the development of our solution, we trained several different models using a consistent training process, referred to as the global training process. The goal of this process is to train models that, at inference time, take an image as input and output text related to that image.

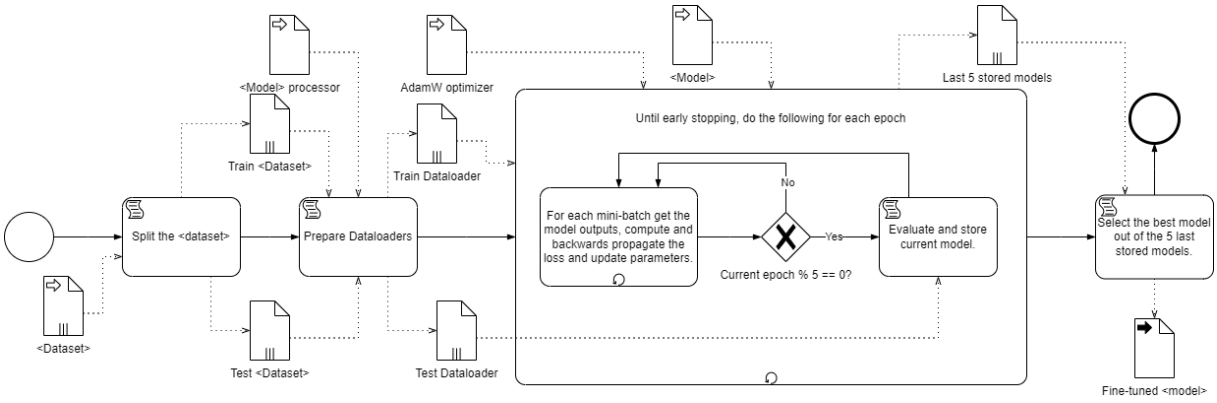



**Figure 5.1:** Our global training process used to train image-to-text models.

Figure 5.1 illustrates the global training process, where placeholders in the format "<>" represent dataset or model names. The input for this process must be a dataset composed of Image-Text pairs, where the text is related to the image. We begin by splitting this dataset, allocating 10% of the records to the test dataset and 90% to the training dataset.

Next, we prepare dataloaders for both the training and test datasets. This involves encoding each image into a tensor and tokenizing the text using the corresponding model's processor, the label is the input_ids (indices of the sequence tokens in the vocabulary) from the tokenized text. For example, the Pix2Struct processor uses a customized image processor for images and a BERT tokenizer for text, wrapping both in a single processor. After encoding and tokenizing, we divide the dataset into batches of size 2 due to memory constraints. The result is a dataloader that we can iterate through to get batches containing encoded images and tokenized text.

For each epoch, we iterate through each batch, providing the image encoding to the model to obtain the tokenized text output. We then compute the loss and perform backpropagation using the model's "loss" and "backward" functions, updating the parameters with the AdamW optimizer. Every fifth epoch, we evaluate the current model on the test dataset and save it to a queue containing the last 5 models. Training stops when the model's performance worsens after two consecutive evaluations on the test dataset. We then select the best model from the last 5 stored models based on the test dataset evaluation.

### 5.1.1 Machine Learning Techniques

As we have seen during the process description we employed several machine learning techniques. Now we will summarize these techniques:

- Early Stopping: This technique involves monitoring the model's performance on a validation set and halting the training process when the performance begins to degrade, thereby preventing

overfitting. We used this technique with the modification of selecting the best of the last 5 stored models.

- Transfer Learning: By leveraging pre-trained models trained on large datasets and fine-tuning them on a smaller, task-specific dataset, we can enhance performance, especially in scenarios where labeled data is limited. The purpose of our global training process is exactly using transfer learning to develop a fine-tuned model for our task.

- Adaptive Learning Rates: This method involves adjusting the learning rates of each parameter individually based on the magnitude of their gradients and the history of their updates. Adaptive learning rates facilitate faster convergence and enhance training stability. The AdamW optimizer, that we used on our training process, employs this tecnhique, we start with a learning rate 1e-5 and let the optimizer adapt it according to the gradient.

- L2 Regularization or Weight Decay: This technique adds a penalty term to the loss function, discouraging large parameter values. By doing so, it helps prevent overfitting and improves generalization performance. This technique is also employed by the AdamW optimizer.

- Bias Correction: Bias correction is a method used in statistical analysis and machine learning to adjust estimates or predictions to accommodate systematic errors, referred to as bias. These biases can stem from various factors such as imperfect measurements, flawed assumptions, or sampling errors. This technique is also employed by the AdamW optimizer.

### 5.1.2 Models trained using our global training process

The global training process was designed to fine-tune models trained on tasks similar to ours. We specifically focused on models that relate images to text and can be trained on Image-Text pairs datasets. Two models that stood out during our search were the GiT 2.3.1 and Pix2Struct 2.3.2 models, described in the literature review section. These models, available on Hugging Face [58], required no additional preparation for our training process, as their API facilitated easy integration.

Both GiT and Pix2Struct are visual recognition models. However, Pix2Struct excelled in our subtask of identifying variables on charts 5.2.1.B, which requires optical character recognition (OCR). This success demonstrates the power of transfer learning since, we did not choose the default Pix2Struct model; instead, we selected a version fine-tuned on the TEXTCAPS 2.3.3 dataset, which also consists of Image-Text pairs and involves OCR tasks on different images. This similarity allowed effective knowledge transfer during our fine-tuning, enabling Pix2Struct to achieve excellent results 5.3.4.B.

## 5.2 Sentences generation process

To begin addressing the task of question generation, we start with a naive approach where we train the GiT and Pix2Struct models directly on the questions. This method involves using our global training process to train both Pix2Struct and GiT models on our main dataset 4.3 of Chart-Question pairs, aiming to train them to generate questions based on given charts.

We expect to encounter several challenges with this approach. Firstly, the generated questions may lack creativity since Image-to-Text models like Pix2Struct and GiT are not trained on extensive text datasets compared to models like GPT-3.5. This could result in a strict adherence to the reference questions. Secondly, extracting the correct information from the charts might be problematic since training the models directly on the questions might hinder their ability to accurately extract chart information, such as variable names, leading to generated questions that reference variables not represented in the chart.

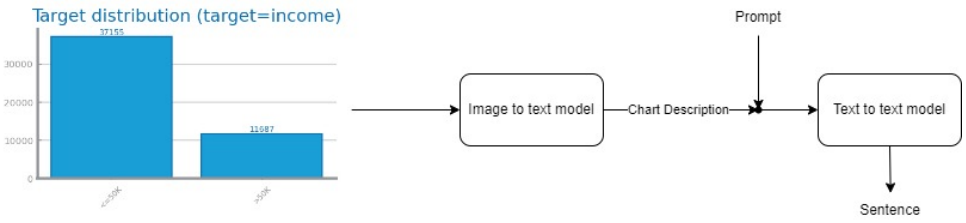



**Figure 5.2:** Our second approach at inference time.

To address these challenges, we propose a second approach outlined in Figure 5.2. This process divides the problem into two distinct tasks. The first task involves generating a caption for a given chart encompassing both the chart type and the variables depicted within it. In the second task, this caption is fed into a text-to-text model to generate the corresponding question.

### 5.2.1 First task: To Generate Charts Captions

For the first task of generating a caption based on a chart, we created the Chart Caption Dataset 4.4.1, consisting of Chart-Caption pairs. We use our global training process to train both Pix2Struct and GiT models on this dataset, resulting in two fine-tuned models for this task.

With this approach we try to tackle the issue of creativity by leveraging a text-to-text model trained on vast and varied datasets encompassing different styles, topics, and contexts. This diversity introduces more variability in the potential outputs, as the model has learned a wide range of patterns and associations in comparison to image-to-text models whose more deterministic nature is reinforced by their training process, which involves directly associating visual features with specific textual descriptions, leading to less variability in their outputs.

Additionally, this approach simplifies the task by focusing on associating a single image with a fixed caption since while our main dataset 4.3 contains 2,630 questions for only 459 charts, while the Chart Caption Dataset 4.4.1 has 459 captions, one for each chart. These captions encapsulate essential information about the chart, including the chart type and the variables represented.

Despite this, we still expect to have challenges. Although most of the caption is fixed according to the chart type, there is a flexible part, which includes the variables represented in the chart (e.g., ”A set of boxplots of the variables [Age, Height, Weight].”), that still poses difficulties. Meaning that the model must first classify the chart type and then identify the variables represented, being a complex task composed of two simpler tasks.

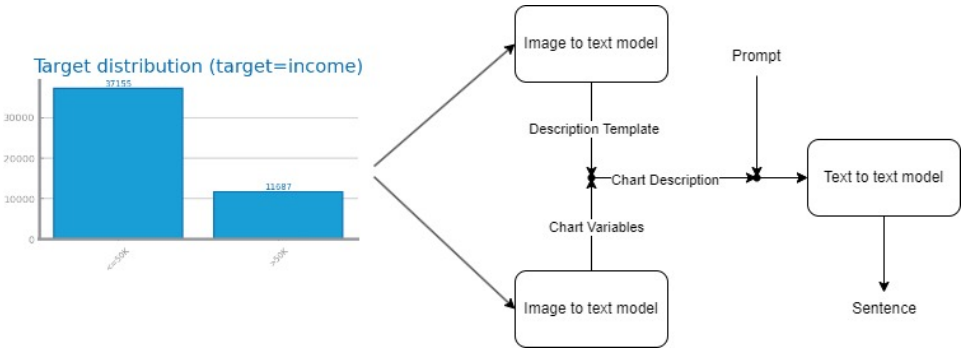



**Figure 5.3:** Our third approach at inference time.

To address these challenges, we propose a third approach depicted in Figure 5.3. This method divides the caption generation task into two sub-tasks. The first sub-task involves classifying the chart type and assigning a description template for that chart type. Each chart type has a unique description template, making this sub-task a classification problem with 15 possible classes. The second sub-task focuses on identifying the variables represented in the chart. We then replace the placeholders in the description template with the identified variables to generate the chart description. This description is then passed as input to the sentence generation model to generate the question.

This approach aims to address previous challenges by training a model specifically for identifying the variables in the chart, which we expect to be a more difficult task than simply classifying the charts, decomposing the complex task of generating a chart caption into two simpler sub-tasks.

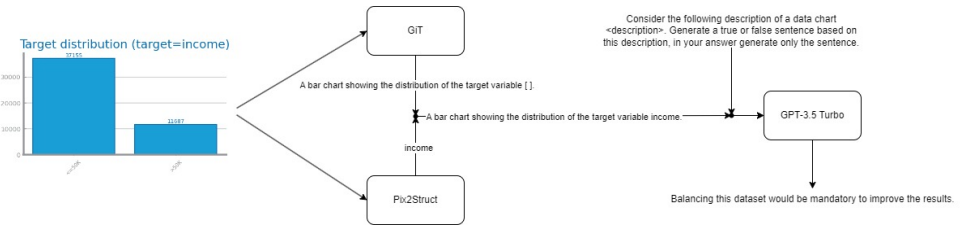



**Figure 5.4:** The final model at inference time.

This last approach proved to be the best performing, as you will see in the results section 5.3.4.D. Figure 5.4 illustrates the components of the final model. The GiT model, fine-tuned for the chart classification sub-task, generates the caption template while the Pix2Struct model, fine-tuned for the variable identification sub-task, generates the variable list, these two components are then joined to form the caption, which in turn is integrated in a prompt and passed to GPT-3.5 Turbo fine-tuned on the sentence generation task that generates the sentence.

#### 5.2.1.A First sub-task: Classifying charts

To address the task of classifying the charts we created an auxiliary dataset composed of Chart-CaptionTemplate pairs, the Caption Templates Dataset 4.4.2.

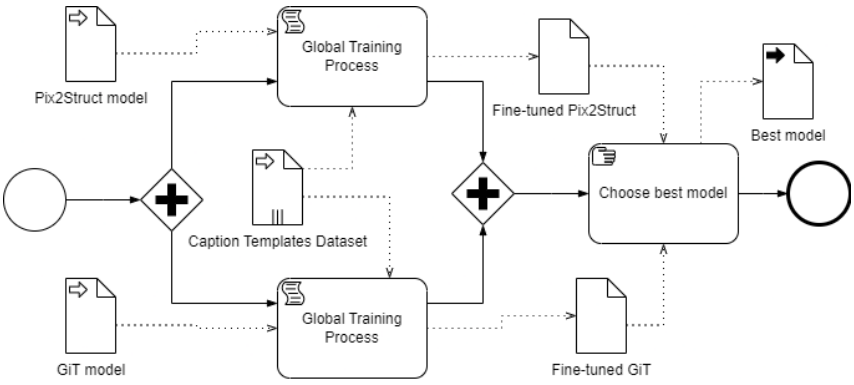



**Figure 5.5:** Training process of the model.

The process to train a model for this task is shown in figure 5.5, we fine-tune both the Pix2Struct and GiT models in the Caption Templates Dataset using our global training process and we select for our final solution the one that performs better.

We expect both fine-tuned models to perform well on this task since it is a classification problem with only 15 possible caption templates and each chart type has distinct characteristics (e.g., a set of boxplots vs. a set of histograms). Even similar chart types, such as multi-line charts 4.2, have different titles (e.g., ”Decision Tree overfitting study” vs. ”KNN overfitting study”), making them easily distinguishable.

#### 5.2.1.B Second sub-task: To identify variables on charts

For the second sub-task we also created an auxiliary dataset, the Chart Variables Dataset 4.4.3 where each record is comprised of the chart and a list of variables represented in the chart.

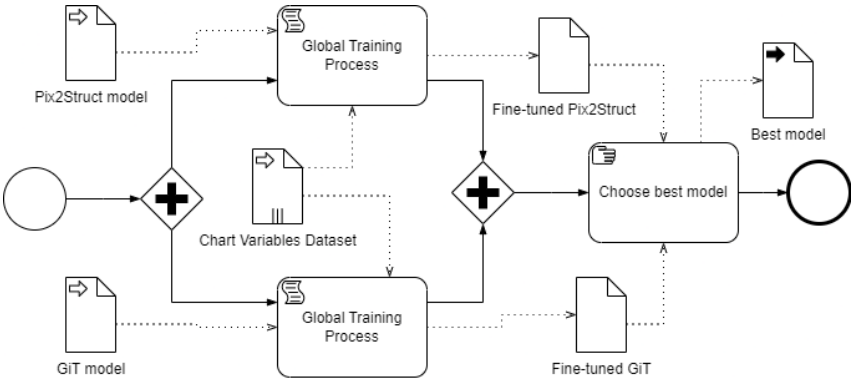



**Figure 5.6:** Training process of the model.

We used a similar process to train this model, as shown in figure 5.6, where we fine-tune both the Pix2Struct and GiT models using the global training process, the difference is that we provide as input to the global training process the Chart Variables Dataset instead of the Caption Templates Dataset.

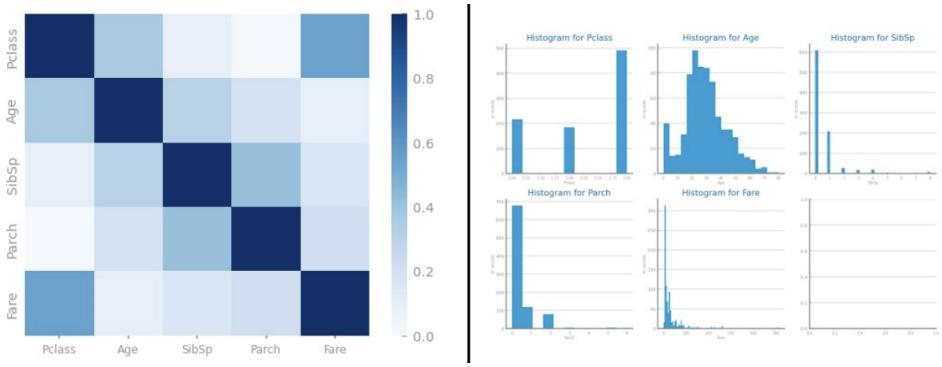



**Figure 5.7:** A comparison between a correlation heatmap and a set of histograms. As we can see the variables names within each chart are in different positions.

This sub-task is expected to be more challenging than the previous one. Identifying all variables represented in the chart requires the model to navigate each chart type and correctly read the variable names from different locations. For example, variable names in a correlation heatmap are positioned differently compared to a set of histograms, as shown in Figure 5.7. The model must learn to extract relevant information without including unnecessary information like the plot title.

Given this challenge, we expect the Pix2Struct model to perform better in this task, as it has been trained on the TEXTCAPS dataset 5.1.2, which also requires identifying text in various positions within different images.

### 5.2.2 Second task: To generate sentences from captions

Now that we have our captions, the next step is to generate the questions given the captions. To accomplish this, we tested two text-to-text models: one large model, GPT-3.5 Turbo 2.3.4, and one smaller model with only 7 billion parameters, Mistral-7B-Instruct-v0.2 2.3.5.

With each model, we explored two approaches. In the first approach, known as zero-shot, we directly queried the model to generate the sentence without fine-tuning. We provided a maximum of three examples in the prompt itself. The prompt for the zero-shot approach is structured as follows:

- system: ”You are a data science teacher creating exam questions.”
- user: ”Consider the following caption of a data chart [caption].”
- assistant: ”I understand, the data chart is [caption].”
- user: ”Generate a true or false sentence based on this caption, in your answer generate only the sentence. As an example consider the following sentences: [listofquestions].”

In the second approach, the fine-tuned approach, the prompt used in this approach closely resembles the one used in the first approach:

- system: You are a data science teacher creating exam questions.
- user: Consider the following caption of a data chart [caption].
- assistant: I understand, the data chart is [caption].
- user: Generate a true or false sentence based on this caption, in your answer generate only the sentence.

To fine-tune this text-to-text models we need to use the required format shown above. So, we constructed a ”dataset” automatically, where each record is the prompt described above with the expected answer added as ”assistant: [question].”

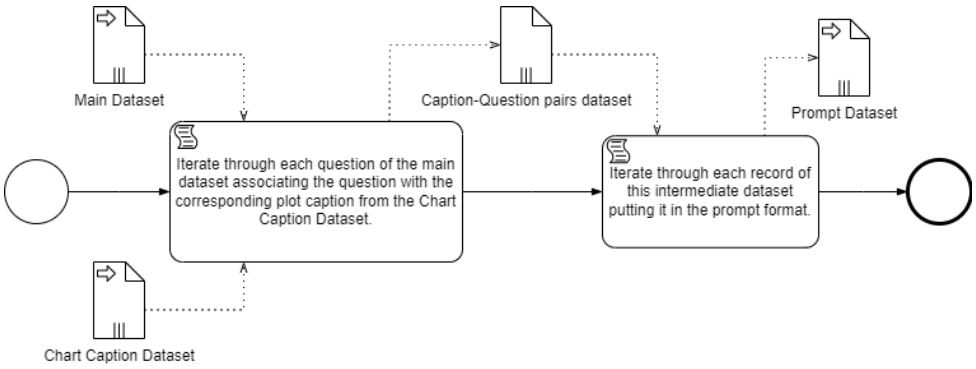



**Figure 5.8:** The process of transforming our data into a prompt format.

As explained before, the main dataset is composed of Chart-Question-Answer triples 4.3, while the Chart Caption Dataset comprises Chart-Caption pairs 4.4.1. In our process (Figure 5.8), we iterate through each question in the main dataset and associate it with the corresponding caption from the Chart Caption Dataset by matching chart names. This leads to intermediary Caption-Question pairs dataset, maintaining the same size as the main dataset (2,630 records). We then iterate through this intermediary dataset, placing the caption and question into the prompt format described earlier 5.2.2. With the prompt dataset ready, we can proceed to describe the training of both models.

### 5.2.3 Mistral-7B fine-tuning

The training process for this model is similar to our global training process, with the difference being that we do not encode any images. Instead, we tokenize the captions. In particular, during the dataloader preparation step, we tokenize each caption and question using the Mistral-7B tokenizer. The label is the input_ids (indices of the sequence tokens in the vocabulary) from the tokenized question. We then divide the dataset into batches of size 2, finishing the preparation of the dataloader. The rest of the process follows the global training process described in Section 5.1.

### 5.2.4 GPT-3.5 Turbo fine-tuning

To fine-tune GPT-3.5 Turbo on our prompt dataset, we used the OpenAI API. The process begins by uploading our prompt dataset to their platform, followed by creating a fine-tuning job with specified parameters. Once fine-tuned, the model can be called via the API to perform chat completions.

## 5.3 Evaluation of Models Performances

All of the models trained on the tasks we have mentioned output some text, so to evaluate our models performance we will need metrics that compare text with a reference text and attribute a score that captures the similarity between both texts. During our research of the visual question generation models that we explained in the literature review section 2.2 we observed that all of them use language generation metrics commonly used for machine translation to evaluate their questions. So we chose these metrics to evaluate our models.

### 5.3.1 Metrics

Before discussing the metrics, let's briefly explain the concepts of precision and recall in this context. Precision measures how many words in the generated sentence are correct with respect to the refer-

ence sentence while recall measures how many words from the reference sentence are present in the generated sentence.

- **BLEU**: BiLingual Evaluation Understudy (BLEU) [59] measures how well a generated text matches one or more reference texts. Initially used in machine translation, BLEU has been adapted for other natural language generation tasks. This metric compares text snippets against reference texts, scoring based on the presence of specific n-grams. A primary limitation of BLEU is its exclusive reliance on n-gram matching, which overlooks overall syntactic correctness. Despite this, we chose this metric because BLEU is a good baseline metric that focuses on precision, comparing the exact match between the generated text and reference texts. BLEU scores range from 0 to 1, with 1 indicating an exact match and 0 indicating no n-gram matches.
- **METEOR**: Metric for Evaluation of Translation with Explicit ORdering (METEOR) [60] is also a metric used to check the accuracy of machine translation output. It is based on the concept of harmonic mean of unigram precision and recall, where recall weights more than precision. METEOR aligns words in the generated and reference texts, considering stems, synonyms (it incorporates synonyms in its evaluation process by using WordNet, a lexical database of English that groups words into sets of synonyms), and word order. METEOR computes precision, recall, and an alignment score to produce an overall metric. This metric is particularly useful for assessing the overall semantic quality of the generated text, allowing for more creativity with synonyms and word order. METEOR scores also range from 0 to 1, with 0 representing no overlap and 1 representing perfect overlap.
- **ROUGE**: Recall-Oriented Understudy for Gisting Evaluation (ROUGE) [61] is a set of metrics used for evaluating the generated text quality against the reference. The metric is primarily used in summarization and text generation tasks. ROUGE-N evaluates the overlap of n-grams between the generated text and the reference text. ROUGE-L is used to evaluate the longest common subsequence in the text sequence. It considers both recall and precision but still relies on surface-level matching and may not capture deeper semantic similarities. We chose ROUGE because it measures how much of the reference text is captured by the generated text, ensuring that important information is not missed. ROUGE scores also range from 0 to 1, with 0 representing no overlap and 1 representing perfect overlap.

To summarize with each metric measures the following:

- **BLEU**: Focuses on precision, ensuring the generated text uses the correct words and phrases found in the reference text.
- **METEOR**: Emphasizes recall, ensuring the generated text captures the important parts of the reference text.

- **ROUGE**: Balances precision and recall, also considering synonyms and stemming, providing a more semantic evaluation.

### 5.3.2 First attempt: Naive approach

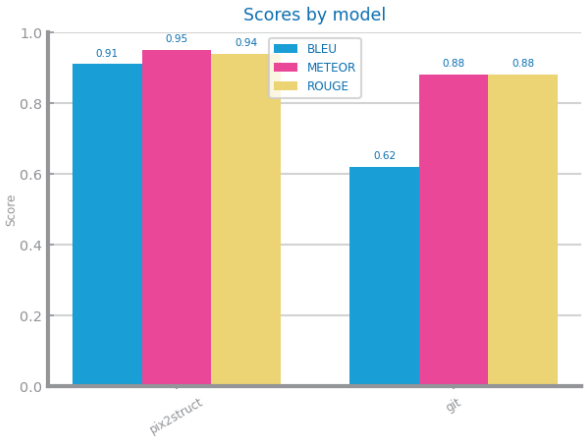



**Figure 5.9:** Results of the GiT and Pix2Struct model when trained directly on the questions dataset.

The outcomes of our naive approach 5.2 were promising, as both models demonstrated the ability to discern the type of chart provided as input and generate corresponding sentences using predefined templates. In Figure 5.9, we present the results obtained for both models, with the Pix2Struct model exhibiting slightly superior performance on the METEOR and ROUGE metrics with 0.95 and 0.94 and a much better performance on the BLEU metric with 0.91 when compared to GiT's 0.62. This can be attributed to the Pix2Struct model's capability to recognize text within the charts, something that the GiT model lacked.

While the results appear promising, two significant challenges persist. Firstly, the generated sentences lack creativity, as the models strictly adhere to generating sentences identical to the reference ones. Secondly, the models fail to identify the variables represented in each chart, leading to sentences with variable names that are not represented in the chart.

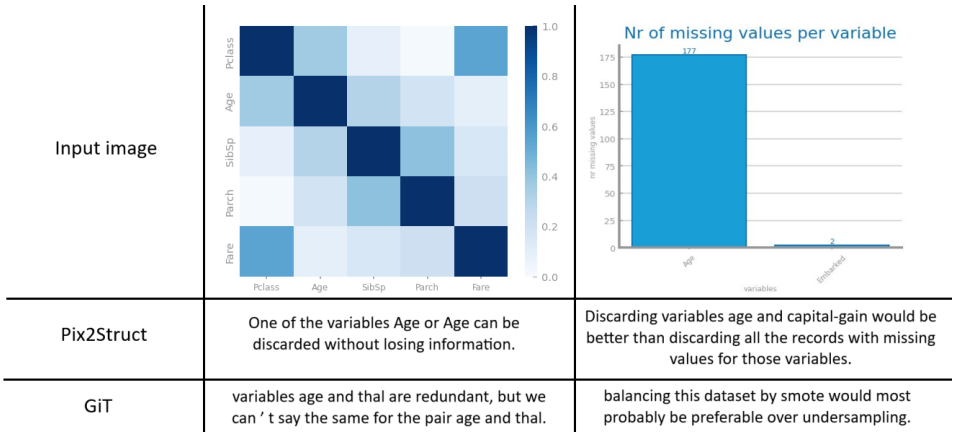



**Figure 5.10:** Examples of sentences generated by both models trained directly on the reference sentences. We can see that the models did not learn how to identify the variables in the chart and is only using the ”age”, the most common variable it found during training.

This issue is illustrated by the examples provided in Figure 5.10, if we look at the questions generated by the Pix2Struct model we see that in the first question it mentions variable ”Age” two times and in the second questions it mentions the variable ”age” and ”capital-gain” that is not represented in the chart this shows that the model did not learn how to extract variable names from the chart since it is mentioning variables that are not represented and the variable ”Age” a lot of times likely due to being the most common variable name on the questions of the training dataset. The first question generated by GiT shows a similar pattern where it used variable ”age” and a variable not represented in the chart.

This issue is illustrated by the examples in Figure 5.10. In the questions generated by the Pix2Struct model, we observe that the first question mentions the variable ”Age” twice, and the second question includes variables ”age” and ”capital-gain,” with ”capital-gain” not represented in the chart. This indicates that the model has not effectively learned to extract variable names from the chart, often mentioning variables that are not present and overusing ”Age” due to its frequency in the training dataset. A similar pattern is seen in the first question generated by the GiT model, which also mentions ”age” and another variable not represented in the chart.

### 5.3.3 Second attempt: Dividing the problem into two tasks

In this attempt, we divided the question generation problem into two tasks: caption generation and sentence generation from captions. This section presents the results of training the Pix2Struct and GiT models on the Chart Caption Dataset to address the caption generation task, as explained in Section 5.2.

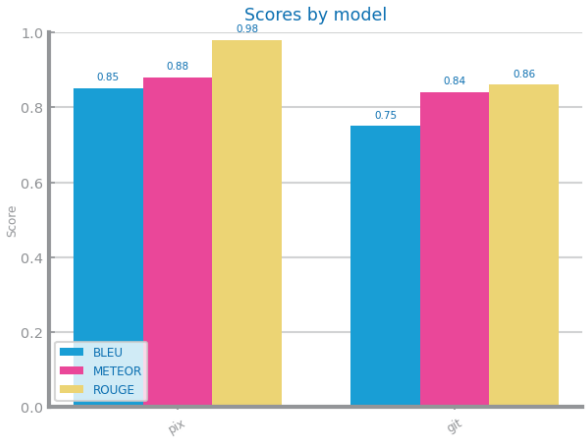



**Figure 5.11:** Results of the GiT and Pix2Struct model when trained on the image captions dataset.

Initially the results of this caption generation task may seem comparable to the first attempt of directly generating questions, as seen in Figure 5.11. The Pix2Struct model achieved scores of 0.85 for BLEU, 0.88 for METEOR, and 0.98 for ROUGE, while the GiT model scored 0.75 for BLEU, 0.84 for METEOR, and 0.86 for ROUGE.

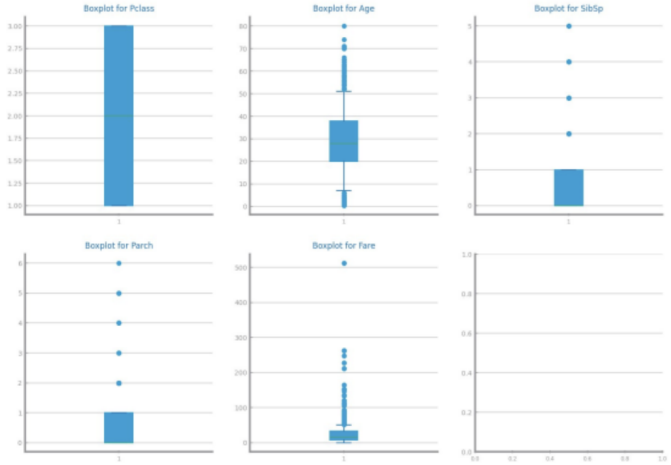
| | |
|---|---|
| Input Image |  |
| Pix2Struct | A set of boxplots of the variables [Pclass, Age, SibSp, Parch]. |
| GiT | a set of boxplots of the variables ['joiningyear ','paymenttier ','age ','experienceincurrentdomain']. |

**Figure 5.12:** Examples of captions generated by both models trained directly on the images captions dataset. We can see that the GiT model did not learn how to identify the variables in the charts, while the Pix2Struct model learned how to identify some variables.

However, if we delve deeper and manually examine the generated captions we observe that the Pix2Struct model demonstrated the ability to identify certain variables depicted in the charts. As shown in Figure 5.12, the Pix2Struct model identified 4 out of the 5 variables in the chart. However, the GiT model did not exhibit this capability and remained unable to recognize variables within the charts.

### 5.3.4 Final process

While the Pix2Struct model demonstrated promising results it still failed in a considerable amount of instances, as per the BLEU metric. This metric gauges the alignment of n-grams between the machine-generated sentences and their reference counterparts. Consequently, the core challenge still persisted in accurately identifying the variables depicted within the charts. To address this, we opted to streamline the task by dividing the generation of chart captions into two sub-tasks. We trained two separate models: one aimed at identifying the variables present in the chart, and the other focused on associating the chart with a caption template. This process was described in section 5.2.1.

#### 5.3.4.A Results for the first sub-task: charts classification

Given that this task essentially amounts to a multiclass classification problem 5.2.1.A, with each caption template serving as a class, we will assess the performance of both models based on the accuracy metric.

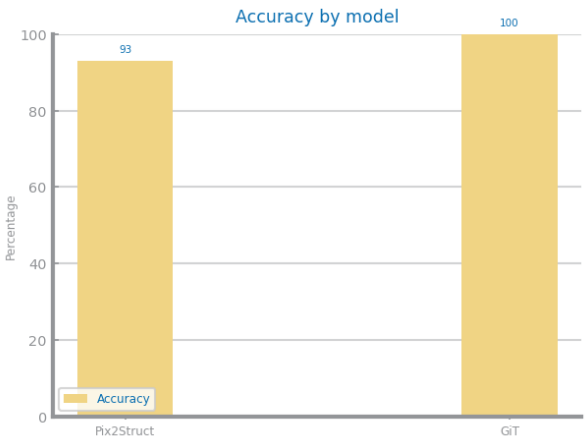



**Figure 5.13:** Results of the GiT and Pix2Struct model when trained on the caption templates dataset.

In figure 5.13 we can see that both models achieved excellent results with the GiT model obtaining a perfect accuracy score and Pix2Struct achieving a score of 0.93. Although both models excelled in the task, we opted for the GiT model due to its slightly superior performance on certain inputs. Specifically, the GiT model demonstrated a better ability to discern the distinctions between random forests and gradient boosting charts, particularly in multi-line charts.

#### 5.3.4.B Results for the second sub-task: identification of variables on charts

Now, let's move on to the second sub-task: identifying chart variables as explained in section 5.2.1.B. While evaluating this task, we considered all three metrics 5.3.1: BLEU, ROUGE, and METEOR. However, METEOR's relevance was diminished as it accounts for linguistic nuances such as synonyms and word order, which are not pertinent to our objective in this particular task. For example, if we have a chart with the variables ["Age", "Length"], we do not want to consider the synonym "Span" as correct for "Length" (i.e., ["Age", "Span"]). On the other hand, word order does not matter in this task, so ["Age", "Length"] and ["Length", "Age"] are both correct. Therefore, in the subsequent discussion, we focused primarily on BLEU and ROUGE metrics, which do not consider synonyms or word order.

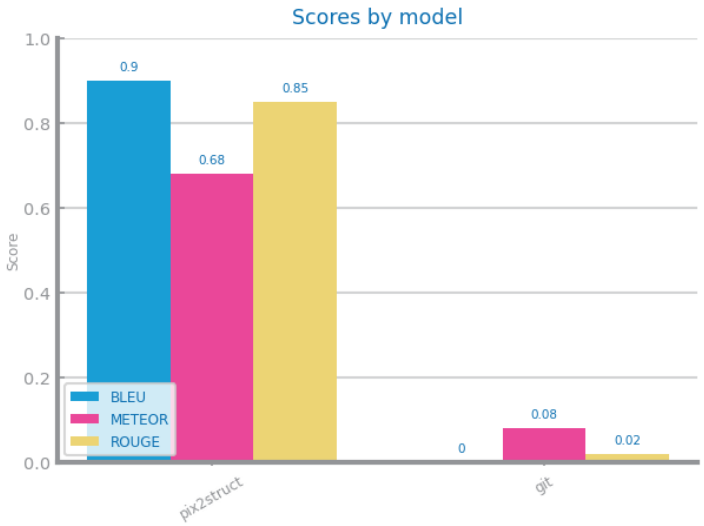



**Figure 5.14:** Results of the GiT and Pix2Struct model when trained on the chart variables dataset.

The performance of both models is illustrated in Figure 5.14. The performance of GiT was underwhelming, while for the Pix2Struct model, at first glance it may seem that the results did not improve from the previous section. However, this isn't the case. Previously, we assessed the complete generated and reference captions, inflating the metrics, as it's easier to generate the caption template than to identify the variables. For instance, consider the comparison of two captions:

- Generated: "A set of histograms of the variables ['Age', 'pH', 'Specific Gravity']."
- Reference: "A set of histograms of the variables ['Age', 'pH']."

The BLEU score between the generated and the reference caption is 0.70. However, if we only consider the list of variables, the BLEU score drops to 0.37, showing a significant decrease.

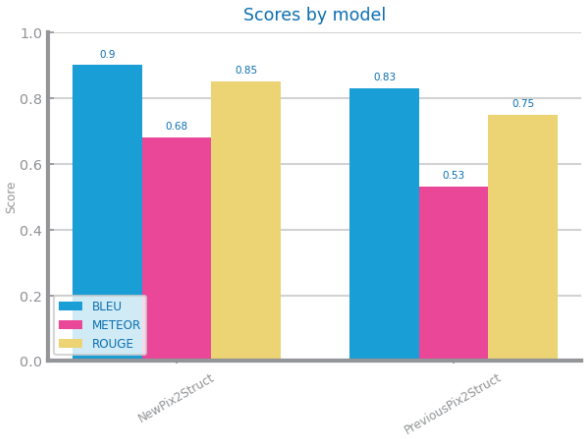



**Figure 5.15:** Results of the new Pix2Struct model compared to the previous attempt.

In image 5.15, we present the true comparison of variable identification between the previous attempt and the current one, highlighting the improved performance of the Pix2Struct model in successfully identifying most variables, with a 7% increase in BLEU and a 10% increase in the ROUGE metric.

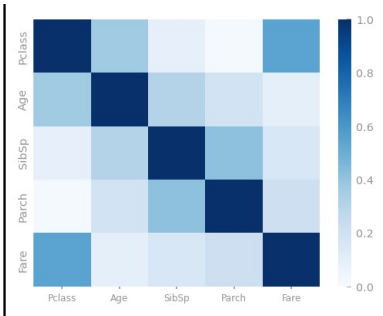
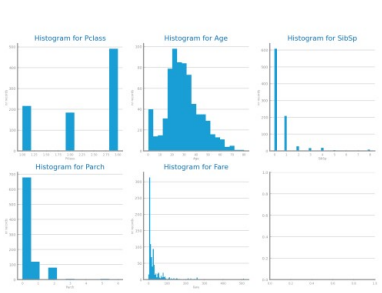
| | | |
|---|---|---|
| Input Image |  |  |
| Pix2Struct | Pclass, Age, SibSp, Parch, Fare | Pclass, Age, SibSp, Parch, Fare |
| GiT | class, age, height, height, weight, height, height, weight, height, weight, weight, height, weight, weight, weight, weight, weight, weight, | pregnancies, gender, age, height, height, weight, height, height, weight, height, weight, height, weight, weight, weight, weight, weight, weight, |

**Figure 5.16:** Examples generated by both models trained on the Chart variables dataset. We can see that the GiT model did not learn how to identify the variables in the charts.

An exemplary outcome is depicted in Figure 5.16, illustrating how Pix2Struct's training approach, which initially prioritizes reading before delving into image structure representation, allowed for a superior performance in this task, primarily through transfer learning.

However, two particular types of charts remain challenging for the model to identify accurately: bar charts for symbolic variables and bar charts for missing values for each variable. Notably, these chart types are underrepresented in our dataset, with 20 and 6 examples, respectively, compared to other charts, which have 33 examples each. Thus, we attribute this phenomenon to a lack of training data.

#### 5.3.4.C Results for sentences generation

Now let's move on to the task of sentence generation. As a quick reminder, we used two approaches for this task: the zero-shot approach and the fine-tuned approach.

In the zero-shot approach, we used Mistral and GPT-3.5 Turbo to generate questions without any prior training. In the fine-tuned approach, we trained both Mistral and GPT-3.5 Turbo on a dataset composed of Caption-Question pairs. For a more detailed explanation of this process, see section 5.2.2.

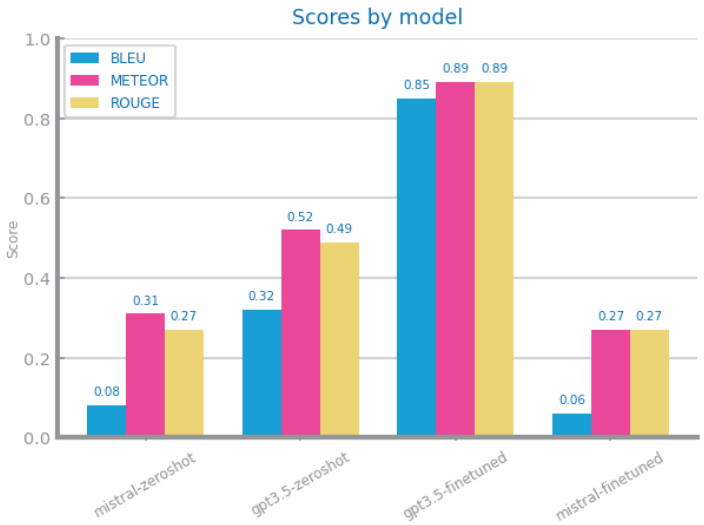



**Figure 5.17:** Results of the mistral and GPT-3.5 Turbo models using both approaches.

In analyzing the results (refer to Figure 5.17), it becomes clear that fine-tuned GPT-3.5 Turbo achieved the best performance overall (BLEU: 0.85, METEOR: 0.89, ROUGE: 0.89). The zero-shot version of GPT-3.5 Turbo followed behind (BLEU: 0.32, METEOR: 0.52, ROUGE: 0.49), while the Mistral model did not yield satisfactory results in either the zero-shot (BLEU: 0.08, METEOR: 0.31, ROUGE: 0.27) or fine-tuned (BLEU: 0.06, METEOR: 0.27, ROUGE: 0.27) approaches.This discrepancy may be attributed to the reduced number of parameters in the Mistral model compared to GPT-3.5 Turbo, suggesting that the model offers greater potential for leveraging transfer learning.

#### 5.3.4.D Global results

Now let's see what were the results for the final_model, the final_model is the model where we assemble all the best performing models in all the previous tasks. The model is described in 5.2.1.

To evaluate the final model, we employed the same test dataset used for evaluating the sentence generation models. However, instead of using the caption, we used the chart itself as input. This approach allows for a direct comparison between the final model and the previous ones. The results are depicted in Figure 5.18.

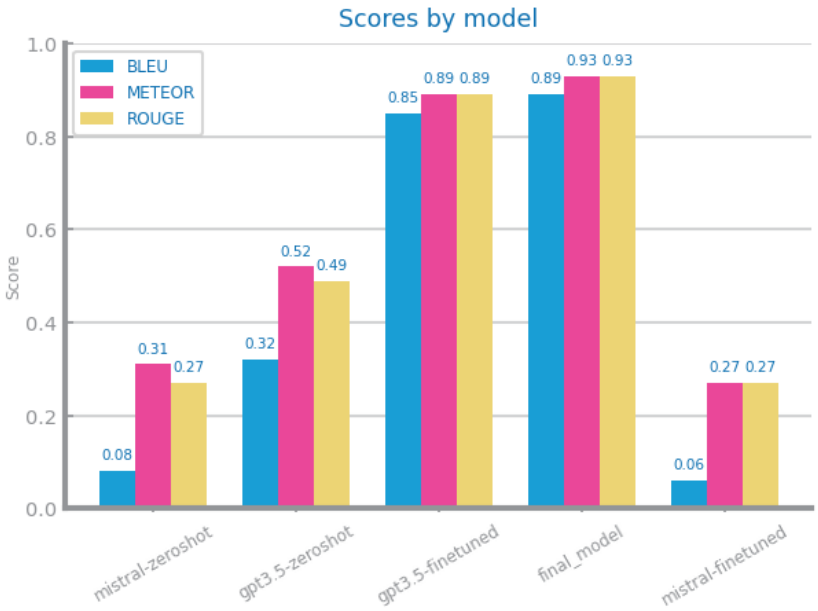



**Figure 5.18:** The final model results compared to the previous models.

The results indicate that the final model performed well, even surpassing the GPT3.5 fine-tuned model. Although initially confusing, it's crucial to note that the reference sentences employed for evaluating the model were manually crafted using templates, incorporating numerous possible combinations of values to fill them. Hence, slight variations in scores between both models may occur, as they might generate correct sentences not present in the reference set. To account for this, we conducted a manual evaluation, the details of which will be discussed in the subsequent section.

## 5.4 Expert evaluation

Although metrics serve as a useful tool for quantifying model performance, they primarily measure the similarity between the generated text and the reference text. Consequently, they may not accurately assess or even penalize the creativity exhibited in the generated sentences. To address this limitation and provide a more comprehensive evaluation of creativity while maintaining sentence quality, we enlisted the expertise of a data scientist to assign a score ranging from 1 to 5 to each generated sentence. In this scoring system, a higher score indicates superior sentence quality. Besides this score we also have two other binary metrics one that says if the sentence is original, or not, to assess the creativity of the model, and the second to see if the generated sentence can be classified as true or false given the input image.

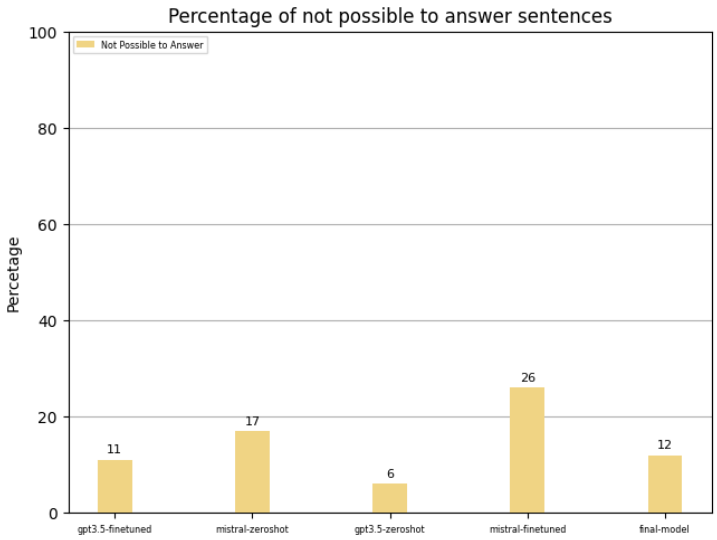



**Figure 5.19:** The percentage of sentences that are not possible to classify as true or false.

In Figure 5.19, we observe the percentage of generated sentences that were deemed not feasible to answer. Both Mistral 7B models underperformed on this metric, aligning with the BLEU, ROUGE, and METEOR scores. Interestingly, the GPT-3.5 fine-tuned model and the final model yielded similar outcomes, which is unsurprising given that the former is a constituent of the latter. However, the GPT-3.5 zero-shot model produced the most favorable results in this regard, with only 6% of generated sentences deemed unanswerable.

Yet, upon closer examination of the sentences generated and labeled by the data scientist, it becomes evident that the zero-shot model tended to produce overly simplistic sentences compared to the final model. This phenomenon can be attributed to the fact that the zero-shot model was not fine-tuned, granting it greater freedom in generating sentences. Consequently, while these sentences may differ from the references, many of them turn out to be too elementary for a data science examination.

For instance, consider the following sentence generated by the zero-shot model: ”A bar chart is a suitable visualization to represent categorical data like species distribution.”. Despite its correcteness, its interest is lacking, being to obvious.

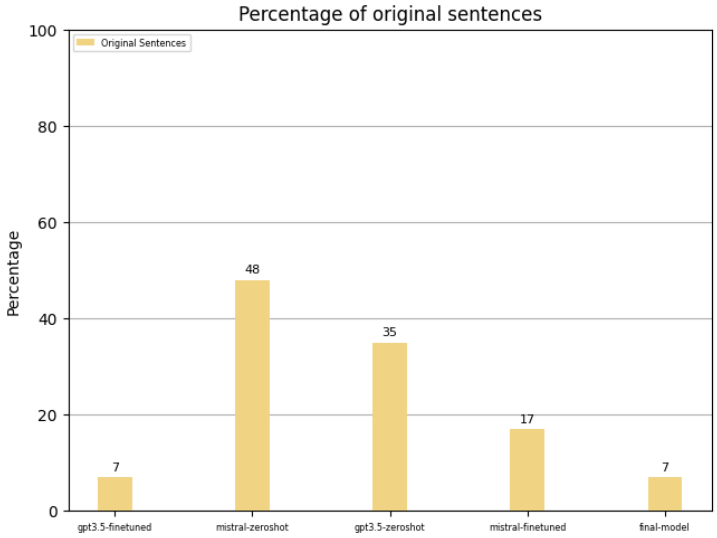



**Figure 5.20:** The percentage of sentences that were classified as original by manual evaluation.

In Figure 5.20, we observe the percentage of sentences deemed original by the data scientist. As anticipated, both zero-shot models exhibited a higher proportion of original sentences compared to the fine-tuned models. Notably, the final model specifically yielded a mere 7% of original sentences.

Nevertheless, it's crucial to acknowledge that a higher percentage of original sentences does not inherently imply superiority. Many of these original sentences tend to be uninteresting or overly simplistic. Hence, there's a delicate balance to strike between fostering creativity and maintaining fidelity to the reference sentences.

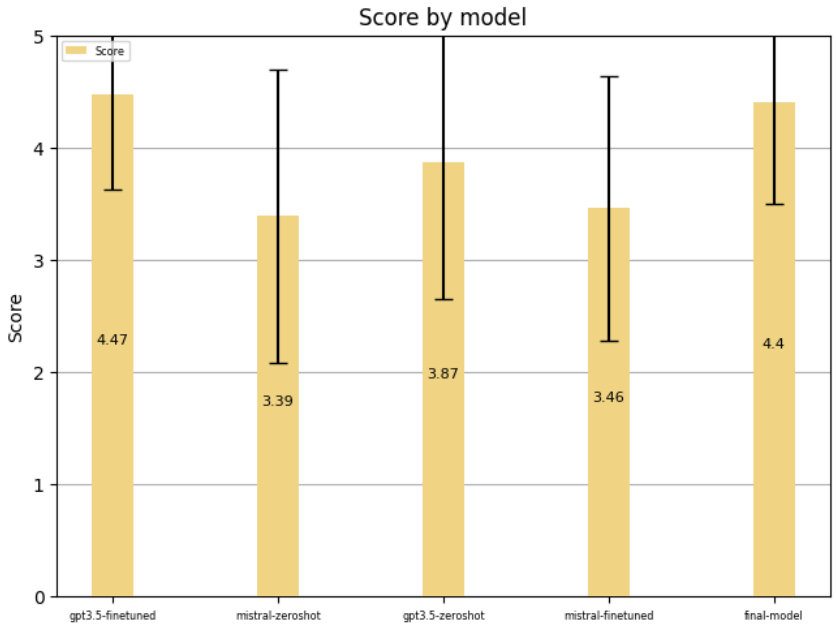



**Figure 5.21:** The data scientist score mean by model, this mean only considers sentences that are possible to answer.

In Figure 5.21, we observe the mean scores across models, calculated exclusively from sentences deemed answerable. Consistent with other metrics, both the final model and the GPT-3.5 fine-tuned one emerged as top performers, exhibiting lower variance in scores indicated by their smaller standard deviations.

This trend can be attributed to several factors. For the zero-shot models, the prevalence of non-interesting sentences likely contributes to their lower mean scores and higher variance. Meanwhile, the Mistral 7B model's inferior performance across all metrics can be attributed to its smaller parameter count compared to GPT-3.5.

Now, let's delve into the distribution of scores for each model and discuss any noteworthy patterns.

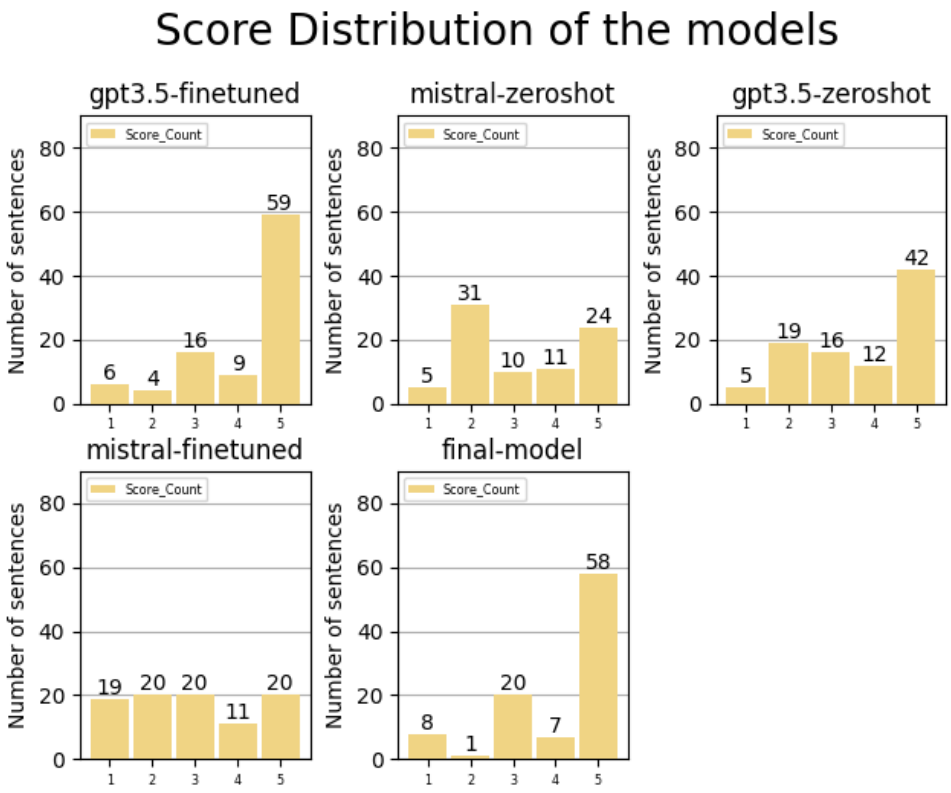



**Figure 5.22:** The distribution of scores of each model.

Now that we have discussed an overview of the manual evaluation we will move on to enter in more detail in the generated sentences and discuss our conclusions and key take aways.

Examining Figure 5.22, we note that both the GPT-3.5 fine-tuned and final models exhibit strong performance, with a majority of generated sentences receiving a score of 5. Interestingly, both models display similar score distributions across all values. This aligns with expectations, considering the fine-tuned model's integration within the final model.

Conversely, the GPT-3.5 zero-shot model demonstrates commendable results, albeit with a notable increase in sentences scored as 2 compared to the fine-tuned model. This disparity stems from the prevalence of non-interesting sentences. A similar trend is observed in the Mistral zero-shot model, indicating a propensity for generating less engaging content.

Meanwhile, the Mistral fine-tuned model presents an even distribution of sentences across all scores, suggesting its struggle to effectively learn the task at hand. This limitation can be attributed to two factors: the model's smaller parameter count compared to GPT-3.5, and the lack of more training data (only 459 records), which compromised its learning capabilities.

Having provided an overview of the manual evaluation, we will now delve deeper into the generated sentences, discussing our conclusions and key insights in greater detail.

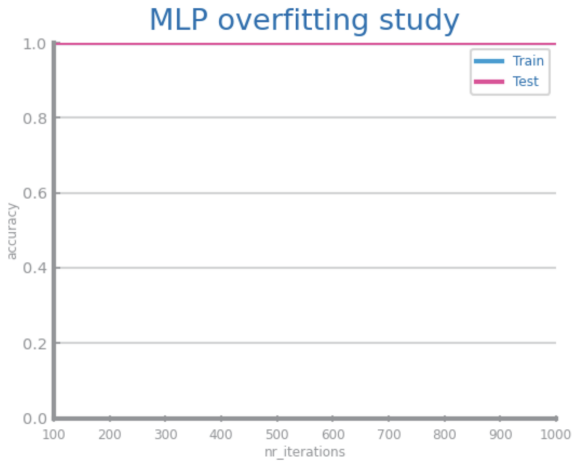
### 5.4.1 Considerations on the generated sentences

In this section, we will delve into the sentences generated by the models. We'll split our discussion into two parts: the first focusing on the sentences generated by the final model, and the second on those generated by the other models.

#### 5.4.1.A Considerations on the sentences generated by the final model

One notable issue we encountered with some of the generated sentences was the misidentification of variables. For instance, the variable "SMK_stat_type_cd" was erroneously labeled as "MK_stat." Such discrepancies were expected, given that the component responsible for identifying variables within the charts does not boast a 100% accuracy rate. To enhance the model's performance in future iterations, refining this variable identification component should be prioritized. This issue is unique to the final model, as the other models under review lacked this variable identification component.

Another challenge surfaced when some of the charts used to generate questions proved to be overly simplistic. Consequently, the resulting questions became too elementary or lacked interest. For example, if a chart depicted consistent 100% accuracy across all episodes in both training and testing data, questions regarding overfitting would invariably yield a simple false answer. Refer to Figure 5.23 for an illustration of this occurrence. The generation of charts, as explained in the data generation section, lacked control measures geared towards educational purposes. Addressing this issue through the automatic generation of informative educational charts in future iterations could yield promising results.




391: We are able to identify the existence of overfitting for MLP models trained longer than 500 episodes.

**Figure 5.23:** An input chart where both the train and test accuracy are 100% for all iterations and the and the sentence generated by the final model.

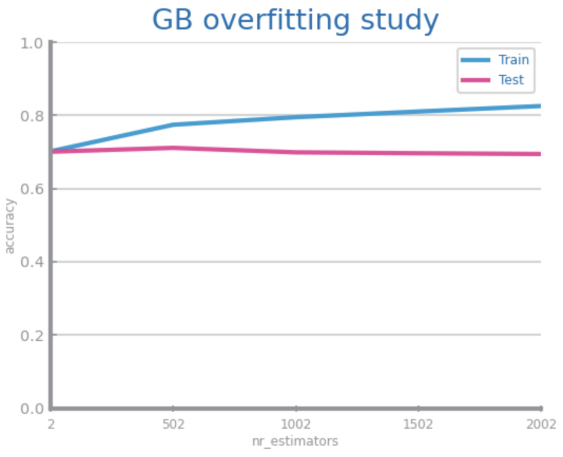
Another intriguing challenge arises with sentences that require more than one chart for an accurate answer. For instance, consider the sentence "Variable skewness seems to be relevant for the majority

of mining tasks," generated from a correlation heatmap as input. While the sentence addresses the relevance of the "skewness" variable, relying solely on the correlation heatmap is insufficient to provide a comprehensive answer; additional charts, such as the decision tree, would be necessary. Therefore, a potential path for future improvement involves modifying the model to accommodate multiple charts as input.

A notable issue we identified stems from grouping binary variables with symbolic variables during dataset generation. Consequently, some sentences originally intended for symbolic variables become overly simplistic when applied to binary variables. For instance, sentences querying if the variable can be seen as ordinal lose their relevance when applied to binary variables, rendering them uninteresting.

Another occurrence, albeit less frequent, involves sentences querying how KNN classifies an expression without providing the value of k (number of neighbors). This indicates a deficiency in the model's training regarding generating sentences about KNN with a decision tree as input. To address this, augmenting the dataset with more examples of such sentences could enhance the model's ability to discern when to include the number of neighbors.

In certain instances of generated sentences, although the sentences themselves attain high scores, the clarity of the accompanying chart is lacking. For example, in Figure 5.24, while the sentence is coherent and can be classified as true or false, the accuracy line exhibits only a slight decrease after 502 estimators. This subtlety could potentially confuse students, as a more pronounced decrease in accuracy would be more conducive to learning. While this aspect deviates from the primary goal of our model, it presents opportunities for future research into automatically generating charts of sufficient quality for educational purposes.




378: We are able to identify the existence of overfitting for gradient boosting models with more than 502 estimators.

**Figure 5.24:** An example of a generated sentence for this chart.

Although the other models boasted higher levels of creativity, as indicated by the percentage of original sentences we discussed earlier, it's worth noting that the final model also demonstrated creativity

while effectively minimizing uninteresting or overly simplistic sentences compared to its counterparts. An example of this creativity is exemplified in the following sentence generated by the final model: ”As we are facing the curse of dimensionality, dummification could be a better option than using binary variables.” This sentence stands as original due to its unique composition, leveraging the model's domain knowledge and insights from the image to generate a novel statement. However, this doesn't imply perfection, as the model occasionally substitutes synonyms for data science terms, which may not always align with the intended context. For instance, it may use 'spread' instead of 'variance,' potentially leading to confusion among students. Hence, striking a balance between creativity and adherence to reference standards is imperative, leaving ample room for improvement in future iterations.

#### 5.4.1.B Considerations on the sentences generated by other models

Beyond the aforementioned considerations, the other models exhibit additional challenges. Firstly, all suffer from the generation of non-interesting sentences, wherein the sentence merely observes the chart without providing meaningful insights. For instance, a sentence like ”All correlations between the variables 'age' and 'waistline' are shown in this heatmap” serves as a prime example. However, such sentences are more prevalent in the Mistral zero-shot model.

Similarly, the Mistral fine-tuned model presented a significant number of non-interesting sentences, albeit fewer compared to its zero-shot counterpart. While this model adhered more closely to reference sentences, it also exhibited errors and occasionally produced nonsensical statements, such as: ”Decision trees with depth = 2 have at most one node at the depth of 3.” This aligns with our expectations, given the model's limited size and inadequate training data.

The GPT-3.5 zero-shot model showcased an intriguing behavior, generating theoretical sentences more frequently than the other models. These theoretical sentences rely solely on domain knowledge and do not necessitate the input chart for interpretation. For instance, ”Overfitting occurs when the training accuracy continues to increase while the validation accuracy starts to decrease.” This behavior was expected since the model lacked fine-tuning to reference sentences. Consequently, it relies more heavily on its own domain knowledge to generate coherent sentences.

In summary, balancing creativity and adherence to reference standards proves challenging, as overly creative models may generate theoretical or overly simplistic sentences. Thus, models that closely align with reference sentences while incorporating a degree of creativity tend to yield better results. However, caution is warranted, as models may substitute data science terms with others that may confuse students. One potential research avenue involves initially training the model on domain knowledge, perhaps using a data science terms dictionary to ensure consistent terminology usage. Additionally, attention to chart generation is crucial, as charts must be tailored for educational purposes to minimize ambiguity in generated sentences, offering ample opportunities for future research in this area.

# 6

# Question Answering

## Contents



To tackle the question-answering task, we decided to use a process similar to the one employed in the question generation task. This decision was made because building a solution from the ground up would require extensive computational resources and data, both of which are limited. Additionally, we already have a process in place for training models to extract information from charts 5.1, a crucial step in solving this task. Therefore, leveraging this existing process provides an excellent starting point.

## 6.1 Training process

In this section, we will provide an overview of the training process leading to the question-answering model and an an overview of the question answering model at inference time.

To tackle this task, we divided the question-answering process into two sub-tasks: data extraction and question answering. In the data extraction task, we generate a caption containing all the necessary information to answer questions about the chart. In the question answering task, given a question and a caption with the chart's information, we classify the question/sentence as true or false.

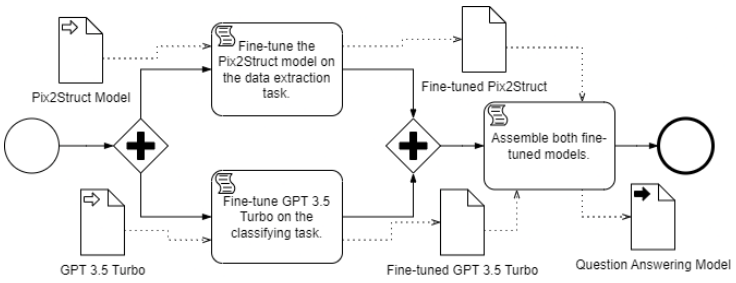



**Figure 6.1:** The global overview of the training process.

The training process, illustrated in Figure 6.1, begins with fine-tuning the Pix2Struct model for the task of extracting data from charts. This process is detailed in section 6.2.1. Next, we fine-tune the GPT-3.5 Turbo model for the question answering task, as described in section 6.3.1. Once both models are fine-tuned, we combine them to create the final question-answering model.

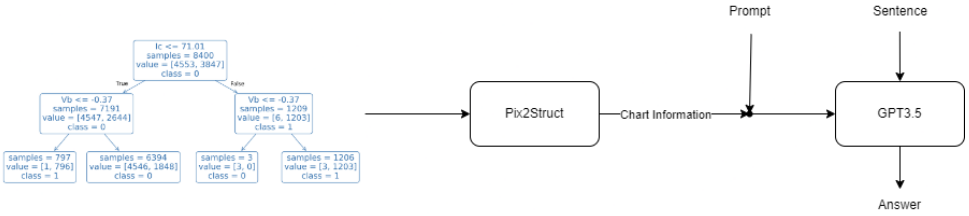



**Figure 6.2:** Question Answering model at inference time.

The assembled question answering model is shown in Figure 6.2. At inference time, the process begins with an input image that is fed into the fine-tuned Pix2Struct model. This model generates a caption containing the chart information. The caption is then formatted into a prompt, as explained in section 6.3.1, and given as input to the fine-tuned GPT-3.5 Turbo model. Along with the prompt, the GPT-3.5 Turbo model also receives the question/sentence and determines whether it is true or false.

## 6.2 Extract Chart Data

Our initial intuition for tackling this task was to reuse the component that generates chart captions from the visual question generation process. However, we encountered a problem with this approach: answering questions requires more detailed information than generating questions. For example, consider the sentence, "Variable Age is balanced.". To generate this question, we only need to extract the variable

name from the chart. However, to classify the sentence as true or false, we need to determine whether the variable ”Age” is actually balanced, which requires extracting more detailed information beyond just the variable names.

### 6.2.1 Process description

The first step in addressing the task of extracting data was to create a new auxiliary dataset consisting of Chart-Data pairs, where ”Data” is a caption encapsulating all the information needed to answer questions about the chart. This auxiliary dataset is referred to as the Chart Data Dataset, explained in section 4.4.4.

With the dataset prepared, we needed to choose a model for fine-tuning. We selected the Pix2Struct model, specifically the version trained on TEXTCAPS, due to its superior performance in the task of identifying variables on charts 5.3.4.B. Given the similarity between these tasks, we determined this model to be the most appropriate choice.

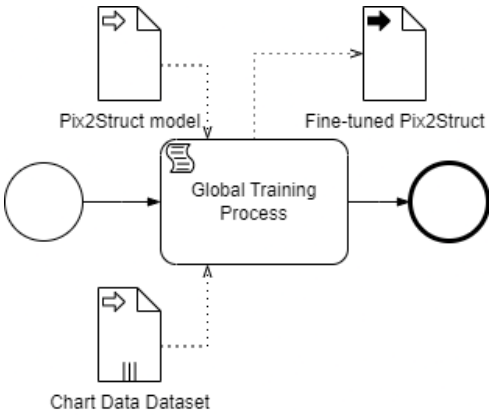



**Figure 6.3:** Training process of the model.

The training process is illustrated in Figure 6.3. Using our Global Training Process described in section 5.1, we provided the Pix2Struct model and the Chart Data Dataset as input and then the Global Training Process outputs the fine-tuned Pix2Struct model.

We expect that this task will yield poorer results compared to the task of identifying variables on charts, as it presents an increased level of difficulty. While the previous task only required identifying variable names, this task necessitates extracting more information from the chart. For instance, consider the sentence, ”Variable Age is balanced.” To generate this question, we only need to extract the variable name from the chart. However, to classify the sentence as true or false, we need to determine whether the variable ”Age” is actually balanced. The details of the information extracted from each chart are detailed in section 4.4.4.

### 6.2.2 Evaluation

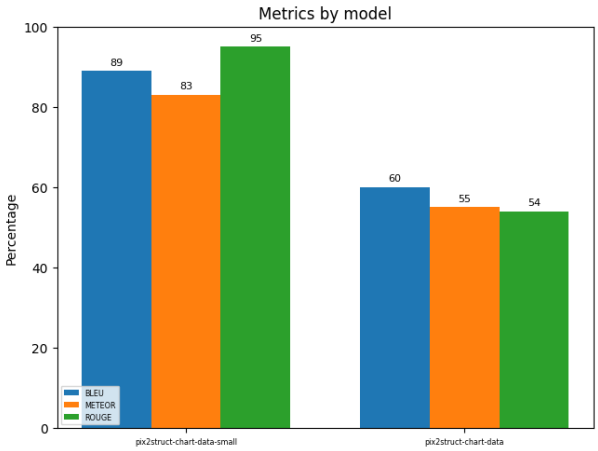



**Figure 6.4:** Results of the models: on the left, the results using only the charts where the model performed better; on the right, the overall results.

To evaluate the results of this task, we used the same metrics as before: BLEU, METEOR, and ROUGE. The model's performance, shown in Figure 6.4, indicates that when considering all types of charts, the results are not satisfactory (BLEU: 0.6, METEOR: 0.55, ROUGE: 0.54). However, if we focus only on charts where the model successfully extracted all the necessary elements (decision trees, PCA histograms, and all bar charts), the results improve significantly (BLEU: 0.89, METEOR: 0.83, ROUGE: 0.95).

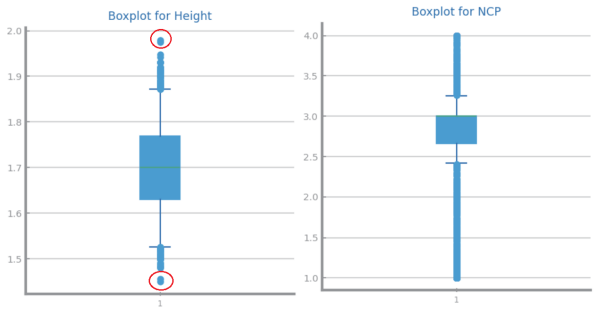



**Figure 6.5:** The encircled dots are outliers. According to our definition, the chart on the right has no outliers.

From a manual evaluation, we discovered that the model struggles with identifying spacing and colors. For instance, in terms of spacing, the model cannot accurately determine whether a boxplot has outliers. Normally, any dots beyond the whiskers would be considered outliers. However, our definition considers a dot as an outlier only if there is a gap between it and the main distribution. This distinction, as shown in Figure 6.5, proved too challenging for the model to learn.

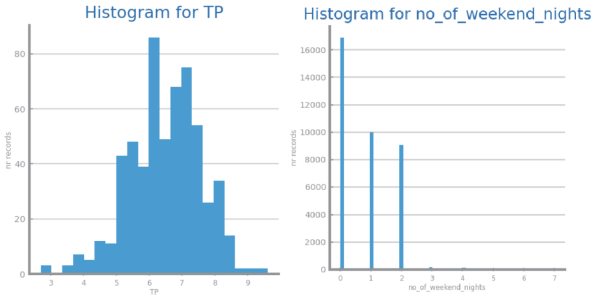



**Figure 6.6:** On the left is a normal histogram; on the right, a histogram with spacing between bars indicating that the variable can be seen as ordinal.

Another spacing issue arises with histograms, where we want the model to detect the spacing between bars to determine if a variable can be seen as ordinal, see Figure 6.6. Detecting the spacing between bars in the right chart could help deduce if the variable is ordinal. However, teaching the pix2struct model to recognize this spacing proved difficult. Similarly, detecting outliers in histograms is also challenging for the same reasons.

The color problem primarily affects the correlation heatmap, where the model cannot accurately differentiate between shades of blue. This is problematic because identifying highly correlated variables is essential for detecting redundancy. A possible solution would be to include numerical values within each square of the heatmap, circumventing this issue.

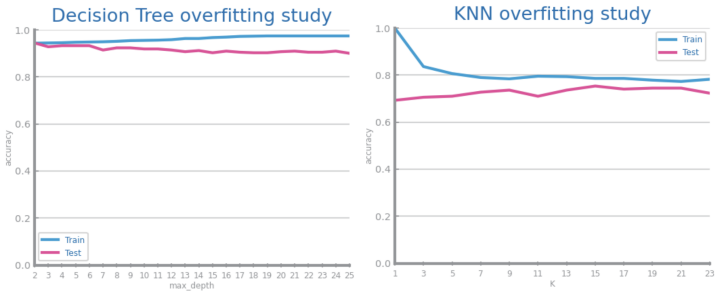



**Figure 6.7:** On the left, a chart with a slight downward slope, making it hard to identify overfitting. On the right, a KNN overfitting study with different visual cues for overfitting.

Lastly, the model also struggles with overfitting charts due to spatial awareness issues. It cannot reliably determine if a chart exhibits overfitting. Some charts have only a slight slope, making it difficult to visually identify overfitting. Additionally, KNN overfitting charts, which overfit with a low amount of neighbors, present a unique challenge since the overfitting curve is different from the other charts. For example, in Figure 6.7, the left chart's slight downward slope complicates overfitting identification, while the right chart shows a KNN overfitting study where visual identification differs from other overfitting studies.

The key takeaway is that the Pix2Struct model faces challenges with tasks requiring spatial aware-

ness. This is due to its pretraining approach, which represents the underlying structure of the image in HTML. While this method is effective for representing visually situated text, it does not perform well with tasks involving spatial awareness and color recognition. This limitation is also highlighted in the model's paper, where the authors acknowledge that Pix2Struct lags behind the state of the art for natural images [17]. However, this architecture excels in chart-related tasks where information extraction relies heavily on visually situated text. Examples include decision trees, histograms with numbers on top of the bars, and bar charts with numbers on top of the bars. As shown in Figure 6.4, the results for these types of charts were significantly better (BLEU: 0.89, METEOR: 0.83, ROUGE: 0.95).

To improve this component, future work could explore other visual models with enhanced spatial awareness and color recognition capabilities. It may also be beneficial to focus the training on one task at a time by creating specific datasets for each problem, such as one for detecting overfitting in charts and another for identifying outliers in boxplots. This approach could address the complexity of learning these tasks simultaneously from a single dataset.

## 6.3 Classifier

The next step is to train a model to classify input sentences as true or false based on the data extracted from the chart. For the model to perform this classification, it needs to have domain knowledge about data science. While several approaches could be employed, such as representing domain knowledge in a graph structure or using a dictionary of definitions, we chose a more generalist approach. We opted to use a model pre-trained on a large corpus of text including a diverse range of topics, like GPT-3.5 Turbo, as its pre-training includes data science concepts necessary for classifying the questions.

### 6.3.1 Process description

We decided to try two approaches similar to generating sentences from captions described in section 5.2.2: the zero-shot and the fine-tuned approach. The model chosen for both approaches was GPT-3.5 Turbo for two main reasons. Firstly, as mentioned earlier, the model was pre-trained on a large corpus of text covering various topics, including data science concepts. Secondly, the model performed better on the question generation task compared to a smaller model, Mistral 7B. This conclusion is also supported by our empirical testing, where GPT-3.5 Turbo showed greater knowledge of data science concepts when compared to Mistral 7B.

The prompt used for both the zero-shot and the fine-tuned approaches was the same and is as follows:

- system: You are a student doing a data science exam.

- user: Consider the following chart [formatteddata], classify the following sentence about the chart as true or false: [sentence].

To fine-tune the GPT-3.5 Turbo model, we created a dataset of prompts using the format described above, with the addition of the answer: ”assistant: [True or False]”.

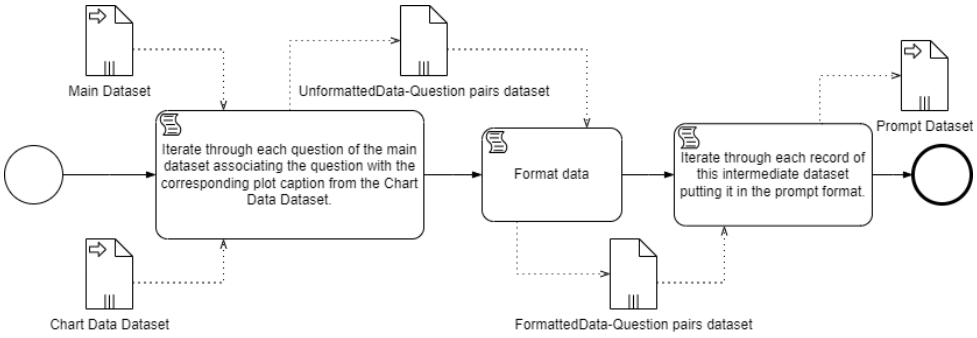



**Figure 6.8:** The process of transforming our data into a prompt format.

Figure 6.8 shows the process of creating this prompt dataset. We start by iterating through each question in the main dataset and associating it with the corresponding caption from the Chart Data Dataset by matching the chart name. Then, we format the caption using a function to transform it from the format described in section 4.4.4 to a more readable format (the detailed transformations for each chart type are in B). We then place the formatted data and the question into the prompt format described earlier 6.3.1.

To fine-tune the GPT 3.5 Turbo model we used the same approach described in section 5.2.4, where we used the OpenAI API to fine-tune the model with this prompt dataset.

### 6.3.2 Evaluation

For evaluation, we used standard classification metrics, since ultimately this is a binary classification task where we need to classify the input sentence as true or false: accuracy, recall, precision, specificity, and F1 score.

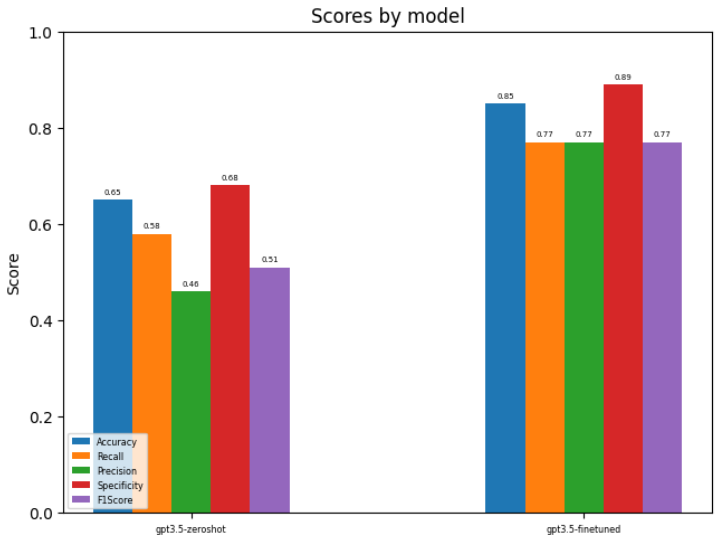



**Figure 6.9:** Results of both the zero-shot and finetuned approaches.

In Figure 6.9 we can see the results of both zero-shot and fine-tuned approaches. We found that the zero-shot model underperformed (Accuracy: 0.65, Recall: 0.58, Precision: 0.46, Specificity: 0.68, F1Score: 0.51). In contrast, the fine-tuned model achieved significantly better results (Accuracy: 0.85, Recall: 0.77, Precision: 0.77, Specificity: 0.89, F1Score: 0.77).

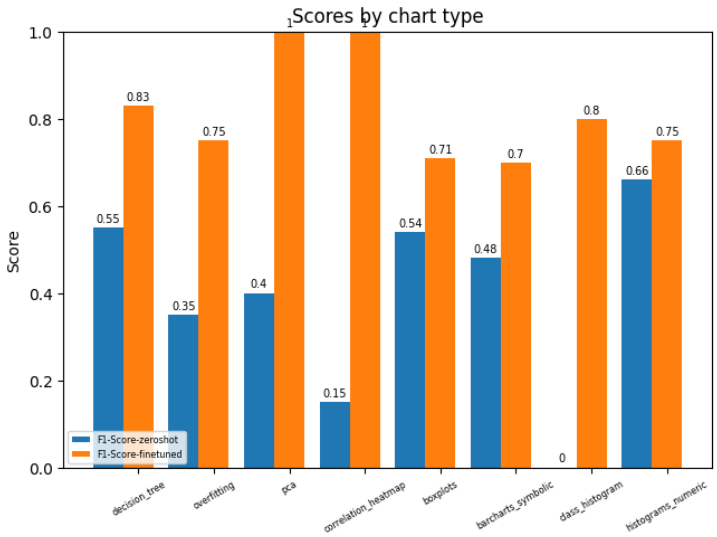



**Figure 6.10:** F1-Score of both the fine-tuned and zero-shot approaches by chart type.

To gain deeper insights and determine whether improvements were consistent across all chart types, we compared the F1-Score for each chart type between both models (Figure 6.10). We chose this metric instead of accuracy because the test dataset was unbalanced, with 105 true labels and 218 false labels. We observed that, for all chart types, the fine-tuned model achieved a higher F1-Score (Decision Trees:

0.83, Overfitting Charts: 0.75, PCA Histograms: 1, Correlation Heatmap: 1, Boxplots: 0.71, Bar Charts: 0.7, Class Histograms: 0.8, Histograms: 0.75) compared to the zero-shot model (Decision Trees: 0.55, Overfitting Charts: 0.35, PCA Histograms: 0.4, Correlation Heatmap: 0.15, Boxplots: 0.54, Bar Charts: 0.48, Class Histograms: 0, Histograms: 0.66). Through a manual analysis, we found the following:

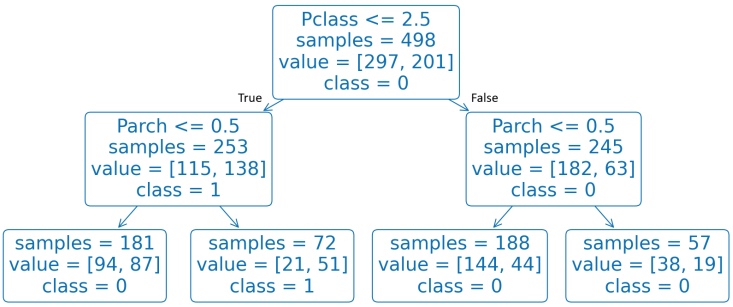



**Figure 6.11:** An example of a decision tree.

- Decision Tree: The zero-shot model knew the necessary formulas but often extracted information incorrectly from the prompt, resulting in poor performance. The fine-tuned model mitigated many of these issues, but still struggled with certain reasoning tasks, such as determining which class the Naive Bayes algorithm would classify for a given set of conditions, lacking the ability to perform the required computations. For instance, consider the decision tree shown in Figure 6.11 and the sentence to be classified: "Considering that $A = True <=> [Pclass <= 2.5]$ and $B = True <=> [Parch <= 0.5]$, it is possible to state that Naive Bayes algorithm classifies (A, not B), as 0.". One of the necessary computations is $P(Pclass \leq 2.5 \mid 0)$. While the correct computation is $\frac{115}{297}$, the model incorrectly computes $\frac{297}{297+201}$. These occasional errors lead to incorrect classifications for some sentences.

- Overfitting Study: The zero-shot model sometimes misinterpreted data from the prompt. Consider both, the sentence "We are able to identify the existence of overfitting for gradient boosting models with more than 1502 estimators." and the chart data "overfitting after 502 estimators." The zero-shot model wrongly classified the sentence as false. The fine-tuned model mostly corrected this but still had issues with sentences like "The random forests results shown can be explained by the lack of diversity resulting from the number of features considered," misclassifying it as true even when the random forest model results were good.

- Principal Component Analysis: The zero-shot model struggled with sentences like "The first 5 principal components are enough for explaining half the data variance." It correctly summed the

explained variances but wrongly classified the sentence as false because the total was not exactly 50%, missing the interpretation that it could be higher. The fine-tuned model corrected this and achieved a perfect score on this type of chart.

- Correlation Heatmap: The zero-shot model performed better but failed when there were no redundant variables, misinterpreting an empty list as having no redundant variables. For example, with the prompt ”A correlation heatmap with 5 features where the redundant pairs are the following: [].” The fine-tuned model corrected this and achieved a perfect score.

- Boxplot: Similar to decision trees, the zero-shot model sometimes extracted incorrect information, leading to wrong classifications. The fine-tuned model reduced this issue but did not eliminate it completely. For example, consider the data: ”{Pclass: {Outliers: F, Balanced: T}, Age: {Outliers: T, Balanced: F}, SibSp: {Outliers: F, Balanced: F}, Parch: {Outliers: T, Balanced: F}, Fare: {Outliers: T, Balanced: F}}”. Given the sentence ”Outliers seem to be a problem in the dataset.” the correct classification should be true since multiple variables have outliers. However, the model classified it as false due to incorrect information extraction.

- Symbolic Bar Chart: The zero-shot model lacked knowledge, such as classifying as false the sentence ”The variable Sex can be seen as ordinal.”, with ”Sex” having values ”male” and ”female”. While not the best encoding, it is true that it can be seen as ordinal. The fine-tuned model improved but still struggled with nuanced sentences, such as differentiating between ”The variable Sex can be seen as ordinal.” (True) and ”The variable Sex can be seen as ordinal without losing information.” (False) misclassifying the latter as True.

- Class Histogram: The zero-shot model lacked knowledge about dataset balancing and confused it with principal component analysis. Fine-tuning mitigated this issue, though it persisted in some cases, possibly due to fewer sentences with this chart type in the dataset.

- Number of Records vs. Number of Variables: Both models performed well, as they understood the concept of the curse of dimensionality. It is also important to note that all records of this chart showed a significant difference between the number of records and the number of variables, making it easier for the models to classify the sentences accurately.

- Numeric Histograms: The zero-shot model performed well overall but struggled with sentences about feature generation, such as ”Feature generation based on the use of variable sc_h wouldn’t be useful, but the use of battery_power seems to be promising.”. The fine-tuned model corrected the classification of these feature generation sentences, however, it did not improve on the other sentences, improving accuracy but not recall, which remained slightly lower than the zero-shot model.

While the fine-tuned model outperformed the zero-shot model, it is still far from perfect, struggling with some reasoning tasks and data extraction from the prompt.

## 6.4 Full evaluation

With all the necessary components in place, we can now assemble the final model using the fine-tuned Pix2Struct for extracting chart data and the fine-tuned GPT-3.5 for classifying sentences. The results of the final model are shown in Figure 6.12.

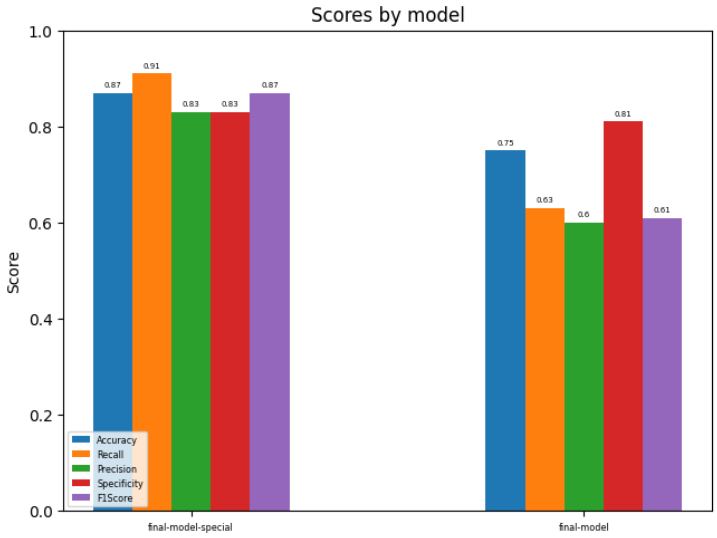



**Figure 6.12:** Results of the final model with all types of charts (on the right) and using only the charts where the Pix2Struct model performed better (on the left).

While the final model achieved good results in accuracy (0.75) and specificity (0.81), it did not perform as well in recall (0.63), precision (0.6), and F1 score (0.61). This is due to the Pix2Struct model's poorer performance on charts requiring spatial awareness. However, when these charts are removed from the evaluation, the results improve significantly across all metrics (Accuracy: 0.87, Recall: 0.91, Precision: 0.83, Specificity: 0.83, F1Score: 0.87), aligning with the fine-tuned GPT-3.5 model's results.

The task of question answering is complex, involving domain knowledge, visual recognition, and reasoning. Consequently, the model we developed is a stepping stone towards more advanced models capable of achieving better results. Several future directions can be pursued:

- Specialized Models: While our model is a generalist approach trained on a dataset of various chart types, future work could involve creating specialized models for each chart type. By representing domain knowledge in a graph, as done in Yang et al. [8], and focusing on each chart type individually, we could achieve more precise control of model reasoning. Combining these specialized

”mini-models” could result in a model that surpasses our current results.

- Visual Component Enhancements: The visual component could also be improved. The fine-tuned Pix2Struct model performs well for charts requiring optical character recognition, but other models could be used for charts requiring more spatial awareness, such as overfitting studies. This targeted approach could enhance performance across different chart types.

These future directions highlight the potential for further development and refinement of models to tackle the intricate task of question answering in data science.

# 7

# Conclusion

The need for online learning personalization and a continuous supply of learning resources demands the automatic generation of learning materials, particularly questions. Despite promising advances in generating textual questions through NLP tools, the results for visual-based questions remain unsatisfactory.

We have developed a pipeline that processes publicly available datasets, transforming them into a convenient form to create data plots and corresponding sentences, which are then manually classified as true or false. These plots and sentences are used to train two models: one that generates new true/false sentences, introducing creativity while maintaining quality, and another that classifies sentences as true or false.

For the question generation task, after achieving promising results with the Pix2Struct and GiT models in extracting information from the charts (Section 5.6) and good results with the fine-tuned GPT-3.5 model (Section 5.7), the architecture produced high-quality sentences with some creativity, as evaluated by a domain expert (university teacher). We observed that balancing creativity and adherence to reference standards proves challenging, as overly creative models may generate theoretical or overly simplistic sentences. Thus, models that closely align with reference sentences while incorporating a degree of creativity tend to yield better results. However, caution is warranted, as models may substitute data science terms with others that may confuse students. One potential research avenue involves

initially training the model on domain knowledge, perhaps utilizing a data science terms dictionary to ensure consistent terminology usage. Additionally, attention to chart generation is crucial, as charts must be tailored for educational purposes to minimize ambiguity in generated sentences, offering ample opportunities for future research in this area.

In the question answering task, after obtaining good results in extracting information from certain types of charts through training the Pix2Struct model (Section 6.1) and good results with the fine-tuned GPT-3.5 model (Section 6.3), the architecture achieved a high degree of accuracy in classifying the sentences, especially for the types of charts where the Pix2Struct model performed well.

For future work, there are many possible directions. These include improving the information extraction from data charts, especially for the question answering task, representing domain knowledge in a graph structure, and developing simpler specialized models for each type of chart that draw from this graph to classify each sentence more accurately. Balancing creativity with adherence to references is another path to explore, as well as developing a structure with data science concepts so that the model generates sentences using correct data science terms. Finally, extending the architecture to generate and answer sentences that require two or more charts is another promising direction. Overall, the tasks of question generation and question answering are complex, involving domain knowledge, visual recognition, and reasoning. Consequently, the architecture we developed is a stepping stone toward more advanced models capable of achieving better results.

# Bibliography


[1] M. W. Gardner and S. Dorling, “Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences,” *Atmospheric environment*, vol. 32, no. 14-15, pp. 2627–2636, 1998.

[2] K. O’Shea and R. Nash, “An introduction to convolutional neural networks,” *arXiv preprint arXiv:1511.08458*, 2015.

[3] A. Wichert and L. Sa-Couto, *Machine Learning — A Journey to Deep Learning with Exercises and Answers*. World Scientific, 2021.

[4] “Long short-term memory (lstm),” https://d2l.ai/chapter_recurrent-modern/lstm.html.

[5] “The encoder–decoder architecture,” https://d2l.ai/chapter_recurrent-modern/encoder-decoder.html.

[6] I. Vasilev, *Advanced Deep Learning with Python: Design and implement advanced next-generation AI solutions using TensorFlow and PyTorch*. Packt Publishing, 2019.

[7] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, “Attention is all you need,” *Advances in neural information processing systems*, vol. 30, 2017.

[8] J. Yang, J. Lu, S. Lee, D. Batra, and D. Parikh, “Visual curiosity: Learning to ask questions to learn visual recognition,” *arXiv preprint arXiv:1810.00912*, 2018.

[9] D. A. Hudson and C. D. Manning, “Gqa: A new dataset for real-world visual reasoning and compositional question answering,” in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 6700–6709.

[10] Y. Li, N. Duan, B. Zhou, X. Chu, W. Ouyang, X. Wang, and M. Zhou, “Visual question generation as dual task of visual question answering,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 6116–6124.

[11] B. N. Patro, S. Kumar, V. K. Kurmi, and V. P. Namboodiri, “Multimodal differential network for visual question generation,” *arXiv preprint arXiv:1808.03986*, 2018.

[12] R. Krishna, M. Bernstein, and L. Fei-Fei, “Information maximizing visual question generation,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 2008–2018.

[13] H. Zhu, R. Togo, T. Ogawa, and M. Haseyama, “A medical domain visual question generation model via large language model,” in *2023 International Conference on Consumer Electronics-Taiwan (ICCE-Taiwan)*. IEEE, 2023, pp. 163–164.

[14] Z. Fan, Z. Wei, S. Wang, Y. Liu, and X.-J. Huang, “A reinforcement learning framework for natural question generation using bi-discriminators,” in *Proceedings of the 27th International Conference on Computational Linguistics*, 2018, pp. 1763–1774.

[15] J. Wang, Z. Yang, X. Hu, L. Li, K. Lin, Z. Gan, Z. Liu, C. Liu, and L. Wang, “Git: A generative image-to-text transformer for vision and language,” *arXiv preprint arXiv:2205.14100*, 2022.

[16] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly *et al.*, “An image is worth 16x16 words: Transformers for image recognition at scale,” *arXiv preprint arXiv:2010.11929*, 2020.

[17] K. Lee, M. Joshi, I. R. Turc, H. Hu, F. Liu, J. M. Eisenschlos, U. Khandelwal, P. Shaw, M.-W. Chang, and K. Toutanova, “Pix2struct: Screenshot parsing as pretraining for visual language understanding,” in *International Conference on Machine Learning*. PMLR, 2023, pp. 18 893–18 912.

[18] O. Sidorov, R. Hu, M. Rohrbach, and A. Singh, “Textcaps: a dataset for image captioning with reading comprehension,” in *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part II 16*. Springer, 2020, pp. 742–758.

[19] A. Q. Jiang, A. Sablayrolles, A. Mensch, C. Bamford, D. S. Chaplot, D. d. l. Casas, F. Bressand, G. Lengyel, G. Lample, L. Saulnier *et al.*, “Mistral 7b,” *arXiv preprint arXiv:2310.06825*, 2023.

[20] D. Bawden, C. Holtham, and N. Courtney, “Perspectives on information overload,” in *Aslib proceedings*, vol. 51, no. 8. MCB UP Ltd, 1999, pp. 249–255.

[21] B. Settles, “Active learning literature survey,” 2009.

[22] R. Moe, “The brief & expansive history (and future) of the mooc: Why two divergent models share the same name,” *Current issues in emerging elearning*, vol. 2, no. 1, p. 2, 2015.

[23] J. S. Drysdale, C. R. Graham, K. J. Spring, and L. R. Halverson, “An analysis of research trends in dissertations and theses studying blended learning,” *The Internet and Higher Education*, vol. 17, pp. 90–100, 2013.

[24] R. E. Mayer, “Thirty years of research on online learning,” *Applied Cognitive Psychology*, vol. 33, no. 2, pp. 152–159, 2019.

[25] G. Kurdi, J. Leo, B. Parsia, U. Sattler, and S. Al-Emari, “A systematic review of automatic question generation for educational purposes,” *International Journal of Artificial Intelligence in Education*, vol. 30, pp. 121–204, 2020.

[26] N.-T. Le, T. Kojiri, and N. Pinkwart, “Automatic question generation for educational applications–the state of art,” in *Advanced Computational Methods for Knowledge Engineering: Proceedings of the 2nd International Conference on Computer Science, Applied Mathematics and Applications (ICCSAMA 2014)*. Springer, 2014, pp. 325–338.

[27] Y. Wang and K. Okamura, “Automatic generation of e-learning contents based on deep learning and natural language processing techniques,” in *Advances in Internet, Data and Web Technologies: The 8th International Conference on Emerging Internet, Data and Web Technologies (EIDWT-2020)*. Springer, 2020, pp. 311–322.

[28] C. A. Nwafor and I. E. Onyenwe, “An automated multiple-choice question generation using natural language processing techniques,” *arXiv preprint arXiv:2103.14757*, 2021.

[29] Y. Chen, W. Li, C. Sakaridis, D. Dai, and L. Van Gool, “Domain adaptive faster r-cnn for object detection in the wild,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 3339–3348.

[30] K. Liu, M. Zhang, and Z. Pan, “Facial expression recognition with cnn ensemble,” in *2016 international conference on cyberworlds (CW)*. IEEE, 2016, pp. 163–166.

[31] Z. Cai and N. Vasconcelos, “Cascade r-cnn: Delving into high quality object detection,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 6154–6162.

[32] B. Kayalibay, G. Jensen, and P. van der Smagt, “Cnn-based segmentation of medical imaging data,” *arXiv preprint arXiv:1701.03056*, 2017.

[33] J. L. Elman, “Finding structure in time,” *Cognitive science*, vol. 14, no. 2, pp. 179–211, 1990.

[34] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[35] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[36] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

[37] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.

[38] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[39] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," *Advances in neural information processing systems*, vol. 27, 2014.

[40] C. Patil and M. Patwardhan, "Visual question generation: The state of the art," *ACM Computing Surveys (CSUR)*, vol. 53, no. 3, pp. 1–22, 2020.

[41] M. Ren, R. Kiros, and R. Zemel, "Exploring models and data for image question answering," *Advances in neural information processing systems*, vol. 28, 2015.

[42] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*. Springer, 2014, pp. 740–755.

[43] D. Klein and C. D. Manning, "Accurate unlexicalized parsing," in *Proceedings of the 41st annual meeting of the association for computational linguistics*, 2003, pp. 423–430.

[44] R. Krishna, Y. Zhu, O. Groth, J. Johnson, K. Hata, J. Kravitz, S. Chen, Y. Kalantidis, L.-J. Li, D. A. Shamma *et al.*, "Visual genome: Connecting language and vision using crowdsourced dense image annotations," *International journal of computer vision*, vol. 123, pp. 32–73, 2017.

[45] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[46] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark *et al.*, "Learning transferable visual models from natural language supervision," in *International conference on machine learning*. PMLR, 2021, pp. 8748–8763.

[47] N. Mostafazadeh, I. Misra, J. Devlin, M. Mitchell, X. He, and L. Vanderwende, "Generating natural questions about an image," *arXiv preprint arXiv:1603.06059*, 2016.

[48] D. Gao, R. Wang, S. Shan, and X. Chen, "Cric: A vqa dataset for compositional reasoning on vision and commonsense," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 45, no. 5, pp. 5561–5578, 2022.

[49] S. Shah, A. Mishra, N. Yadati, and P. P. Talukdar, "Kvqa: Knowledge-aware visual question answering," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 33, no. 01, 2019, pp. 8876–8884.

[50] K. Uehara and T. Harada, "K-vqg: Knowledge-aware visual question generation for common-sense acquisition," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 2023, pp. 4401–4409.

[51] A. Radford, K. Narasimhan, T. Salimans, I. Sutskever *et al.*, "Improving language understanding by generative pre-training," 2018.

[52] B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal *et al.*, "Language models are few-shot learners," *arXiv preprint arXiv:2005.14165*, vol. 1, 2020.

[53] I. E. Allen and J. Seaman, *Changing course: Ten years of tracking online education in the United States.* ERIC, 2013.

[54] S. Dhawan, "Online learning: A panacea in the time of covid-19 crisis," *Journal of educational technology systems*, vol. 49, no. 1, pp. 5–22, 2020.

[55] G. Van Rossum and F. L. Drake, *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace, 2009.

[56] W. McKinney *et al.*, "Data structures for statistical computing in python," in *Proceedings of the 9th Python in Science Conference*, vol. 445. Austin, TX, 2010, pp. 51–56.

[57] "Matplotlib, a python library for visualization," https://matplotlib.org/.

[58] "Hugging face website," https://huggingface.co/.

[59] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "Bleu: a method for automatic evaluation of machine translation," in *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, 2002, pp. 311–318.

[60] S. Banerjee and A. Lavie, “Meteor: An automatic metric for mt evaluation with improved correlation with human judgments,” in *Proceedings of the acl workshop on intrinsic and extrinsic evaluation measures for machine translation and/or summarization*, 2005, pp. 65–72.

[61] C.-Y. Lin, “Rouge: A package for automatic evaluation of summaries,” in *Text summarization branches out*, 2004, pp. 74–81.

# A

# Dataset examples

In this appendix we show records of the datasets used in the development process.

**Listing A.1:** Example of records for the Main Dataset

```
1 Id;Chart;Question
2 0;ObesityDataSet_decision_tree.png;The variable FAF discriminates between the target values, as shown in
      the decision tree.
3 1;ObesityDataSet_decision_tree.png;Variable Height is one of the most relevant variables.
4 2;ObesityDataSet_decision_tree.png;A smaller tree would be delivered if we would apply post-pruning,
      accepting an accuracy reduction of 8%.
5 3;ObesityDataSet_decision_tree.png;As reported in the tree, the number of False Positive is smaller than the
      number of False Negatives.
6 4;ObesityDataSet_decision_tree.png;The specificity for the presented tree is higher than 75%.
7 5;ObesityDataSet_decision_tree.png;The number of True Negatives reported in the same tree is 50.
8 6;ObesityDataSet_decision_tree.png;The number of False Positives is lower than the number of False
      Negatives for the presented tree.
9 7;ObesityDataSet_decision_tree.png;The variable FAF seems to be one of the two most relevant features.
```

**Listing A.2:** Examples of the Chart Caption Dataset

```
1 Chart;Caption
2 ObesityDataSet_overfitting_mlp.png;A multi-line chart showing the overfitting of a mlp where the y-axis
      represents the accuracy and the x-axis represents the number of iterations ranging from 100 to 1000.
3 ObesityDataSet_overfitting_gb.png;A multi-line chart showing the overfitting of gradient boosting where the
      y-axis represents the accuracy and the x-axis represents the number of estimators ranging from 2 to
      2002.
4 ObesityDataSet_overfitting_rf.png;A multi-line chart showing the overfitting of random forest where the y-axis
      represents the accuracy and the x-axis represents the number of estimators ranging from 2 to 2002.
5 ObesityDataSet_overfitting_knn.png;A multi-line chart showing the overfitting of k-nearest neighbors where the
      y-axis represents the accuracy and the x-axis represents the number of neighbors ranging from 1 to 23.
6 ObesityDataSet_overfitting_decision_tree.png;A multi-line chart showing the overfitting of a decision tree
      where the y-axis represents the accuracy and the x-axis represents the max depth ranging from 2 to 25.
7 ObesityDataSet_pca.png;A bar chart showing the explained variance ratio of 8 principal components.
8 ObesityDataSet_correlation_heatmap.png;A heatmap showing the correlation between the variables of the
      dataset. The variables are ['Age', 'Height', 'Weight', 'FCVC', 'NCP', 'CH2O', 'FAF', 'TUE'].
9 ObesityDataSet_boxplots.png;A set of boxplots of the variables ['Age', 'Height', 'Weight', 'FCVC', 'NCP',
      'CH2O', 'FAF', 'TUE'].
10 ObesityDataSet_histograms_symbolic.png;A set of bar charts of the variables ['CAEC', 'CALC', 'MTRANS',
      'Gender', 'family_history_with_overweight', 'FAVC', 'SMOKE', 'SCC'].
11 ObesityDataSet_class_histogram.png;A bar chart showing the distribution of the target variable NObeyesdad.
12 ObesityDataSet_nr_records_nr_variables.png;A bar chart showing the number of records and variables of the
      dataset.
13 ObesityDataSet_histograms_numeric.png;A set of histograms of the variables ['Age', 'Height', 'Weight',
      'FCVC', 'NCP', 'CH2O', 'FAF', 'TUE'].
```

**Listing A.3:** Examples of the Caption Templates Dataset

```
1 Chart;Caption
2 ObesityDataSet_decision_tree.png;An image showing a decision tree with depth = 2 where the first decision is
      made with the condition [] and the second with the condition [].
3 ObesityDataSet_overfitting_mlp.png;A multi-line chart showing the overfitting of a mlp where the y-axis
      represents the accuracy and the x-axis represents the number of iterations ranging from 100 to 1000.
4 ObesityDataSet_overfitting_gb.png;A multi-line chart showing the overfitting of gradient boosting where the
      y-axis represents the accuracy and the x-axis represents the number of estimators ranging from 2 to
      2002.
5 ObesityDataSet_overfitting_rf.png;A multi-line chart showing the overfitting of random forest where the y-axis
```

represents the accuracy and the x-axis represents the number of estimators ranging from 2 to 2002.
ObesityDataSet_overfitting_knn.png;A multi-line chart showing the overfitting of k-nearest neighbors where the y-axis represents the accuracy and the x-axis represents the number of neighbors ranging from 1 to 23.
ObesityDataSet_overfitting_decision_tree.png;A multi-line chart showing the overfitting of a decision tree where the y-axis represents the accuracy and the x-axis represents the max depth ranging from 2 to 25.
ObesityDataSet_pca.png;A bar chart showing the explained variance ratio of [] principal components.
ObesityDataSet_correlation_heatmap.png;A heatmap showing the correlation between the variables of the dataset. The variables are [].
ObesityDataSet_boxplots.png;A set of boxplots of the variables [].
ObesityDataSet_histograms_symbolic.png;A set of bar charts of the variables [].
ObesityDataSet_class_histogram.png;A bar chart showing the distribution of the target variable [].
ObesityDataSet_nr_records_nr_variables.png;A bar chart showing the number of records and variables of the dataset.
ObesityDataSet_histograms_numeric.png;A set of histograms of the variables [].

**Listing A.4:** Examples of the Chart Variables Dataset

```
Chart;Variables
ObesityDataSet_decision_tree.png;FAF, Height
ObesityDataSet_correlation_heatmap.png;Age, Height, Weight, FCVC, NCP, CH2O, FAF, TUE
ObesityDataSet_boxplots.png;Age, Height, Weight, FCVC, NCP, CH2O, FAF, TUE
ObesityDataSet_histograms_symbolic.png;CAEC, CALC, MTRANS, Gender, family_history_with_overweight, FAVC, SMOKE, SCC
ObesityDataSet_class_histogram.png;NObeyesdad
ObesityDataSet_histograms_numeric.png;Age, Height, Weight, FCVC, NCP, CH2O, FAF, TUE
ObesityDataSet_pca.png;8
customer_segmentation_decision_tree.png;Family_Size, Work_Experience
customer_segmentation_correlation_heatmap.png;Age, Work_Experience, Family_Size
customer_segmentation_boxplots.png;Age, Work_Experience, Family_Size
customer_segmentation_histograms_symbolic.png;Profession, Spending_Score, Var_1, Gender, Ever_Married, Graduated
customer_segmentation_mv.png;Ever_Married, Graduated, Profession, Work_Experience, Family_Size, Var_1
customer_segmentation_class_histogram.png;Segmentation
customer_segmentation_histograms_numeric.png;Age, Work_Experience, Family_Size
customer_segmentation_pca.png;3
urinalysis_tests_decision_tree.png;Age, pH
urinalysis_tests_correlation_heatmap.png;Age, pH, Specific Gravity
urinalysis_tests_boxplots.png;Age, pH, Specific Gravity
```

urinalysis_tests_histograms_symbolic.png;Color, Transparency, Glucose, Protein, Epithelial Cells, Mucous Threads, Amorphous Urates, Bacteria, Gender
urinalysis_tests_mv.png;Color
urinalysis_tests_class_histogram.png;Diagnosis
urinalysis_tests_histograms_numeric.png;Age, pH, Specific Gravity
urinalysis_tests_pca.png;3
detect_dataset_decision_tree.png;Ic, Vb
detect_dataset_correlation_heatmap.png;Ia, Ib, Ic, Va, Vb, Vc
detect_dataset_boxplots.png;Ia, Ib, Ic, Va, Vb, Vc
detect_dataset_class_histogram.png;Output
detect_dataset_histograms_numeric.png;Ia, Ib, Ic, Va, Vb, Vc
detect_dataset_pca.png;6

**Listing A.5:** Examples of the Chart Data Dataset

Id;Data;Chart
13;{'Family_Size <= 2.5': {'samples': 1507, 'value': [484,315,267,441], 'class': 'A', 'True': {'Work_Experience <= 9.5': {'samples': 773, 'value': [234,189,145,205], 'class': 'A', 'True': {'samples': 249, 'value': [80,70,47,52], 'class': 'A'}, 'False': {'samples': 524, 'value': [154,119,98,153], 'class': 'A'}}}, 'False': {'Work_Experience <= 9.5': {'samples': 734, 'value': [250,126,122,236], 'class': 'A', 'True': {'samples': 723, 'value': [242,126,122,233], 'class': 'A'}, 'False': {'samples': 11, 'value': [8,0,0,3], 'class': 'A'}}}};customer_segmentation_decision_tree.png
14;no;customer_segmentation_overfitting_mlp.png
15;overfitting after 502;customer_segmentation_overfitting_gb.png
16;no underfitting,no overfitting,'acc<0.8';customer_segmentation_overfitting_rf.png
17;overfitting until 15;customer_segmentation_overfitting_knn.png
18;overfitting after 3;customer_segmentation_overfitting_decision_tree.png
19;[0.96,0.04,0.01];customer_segmentation_pca.png
20;{3,[]};customer_segmentation_correlation_heatmap.png
21;[F,{Age:{Outliers:F,Balanced:F},Work_Experience:{Outliers:F,Balanced:F},Family_Size:{Outliers:F,Balanced:F}}]; customer_segmentation_boxplots.png
22;{Profession:['Engineer', 'Healthcare', 'Executive', 'Marketing', 'Doctor', 'Artist', 'Lawyer', 'Entertainment', 'Homemaker'],Spending_Score:['Low', 'Average', 'High'],Var_1:['Cat_6', 'Cat_4', 'Cat_3', 'Cat_1', 'Cat_2', 'Cat_5', 'Cat_7'],Gender:['Female', 'Male'],Ever_Married:['Yes', 'No'],Graduated:['Yes', 'No']};customer_segmentation_histograms_symbolic.png
23;{'Ever_Married': 50, 'Graduated': 24, 'Profession': 38, 'Work_Experience': 269, 'Family_Size': 113, 'Var_1': 32};customer_segmentation_mv.png

24;[846,759,550,472];customer_segmentation_class_histogram.png

25;[2627,10];customer_segmentation_nr_records_nr_variables.png

---

# B

# Code

```
def formatData(data):
    if data == 'no':
        return 'A multi-line chart that does not suffer from overfitting.'
    elif 'overfitting after' in data and 'acc=' not in data:
        return f'A multi-line chart that shows {data}.'
    elif 'overfitting until' in data:
        return f'A multi-line chart that shows {data} neighbors.'
    elif ',acc' in data and 'overfitting' in data:
        if 'no' in data.split("nderfitting ")[0]:
            underfitting = 'no'
        else:
            underfitting = data.split("nderfitting ")[1].split(",")[0]

        if 'no' in data.split("overfitting ")[0]:
            overfitting = 'no'
        else:
            overfitting = data.split("overfitting ")[1].split(",")[0]
```

```
18
19          if data.split("ac")[1] == 'c>80':
20              acc = 'test accuracy line surpasses than 80%'
21          elif data.split("ac")[1] == 'c<80':
22              acc = 'test accuracy line does not surpass 80%'
23          else:
24              acc = 'test accuracy line stays at 80%'
25          if underfitting == 'no' and overfitting == 'no':
26              return f'A multi-line chart showing the random forest does not have
27              underfitting, does not enter in overfitting and it\'s {acc}.'
28          elif underfitting != 'no' and overfitting == 'no':
29              return f'A multi-line chart showing the random forest is in
30              underfitting until {underfitting} estimators, does not enter
31              in overfitting and it\'s {acc}.'
32          elif underfitting == 'no' and overfitting != 'no':
33              return f'A multi-line chart showing the random forest does not have
34              underfitting, enters in overfitting after {overfitting} estimators
35              and it\'s {acc}.'
36          else:
37              return f'A multi-line chart showing the random forest is in underfitting
38              until {underfitting} estimators, enters in overfitting after {overfitting}
39              estimators and it\'s {acc}.'
40      elif data == 'acc_rec':
41          return 'A multi-line chart showing the train and test accuracy and recall of
42          a decision tree.'
43      elif 'samples' in data:
44          return f'A decision tree with the following configuration:{data}.'
45      elif data[0] == '[' and len(data.split(",")) > 2:
46          return f'A bar chart showing the explained variance ration of each principal
47          components the values are the following:{data}.'
48      elif data[0] == '{' and data[1].isnumeric():
49          return f'A correlation heatmap with {data[1:].split(",")[0]} features where
50          the redundant pairs are the following:{data[:-1].split(",",1)[1]}.'
51      elif data[0] == '[' and (data[1] == 'T' or data[1] == 'F') and data[2] == ',':
52          return f'A set of boxplots that aren\'t normalized, in the following dictionary
53          we can see each one of them and see if they have outliers and are balanced:
54          {data[:-1].split(",",1)[1]}.'
55      elif data[-1] == '}' and data[-2] == ']':
```

```
56          return f'A set of bar charts for each symbolic variable:{data}.'
57      elif data[0] == '{' and data[-2].isnumeric():
58          return f'A bar chart showing the number of missing values for each variable:{data}.'
59      elif data[0] == '[' and len(data.split(",")) == 2:
60          return f'A bar chart with the following values:{data}.'
61      elif 'Ordinal:' in data:
62          return f'A set of histograms one for each variable, in the following dictionary
63          we can see each one of them and see if they have outliers, if they are balanced
64          and if it makes sense for them to be ordinal:{data}.'
```