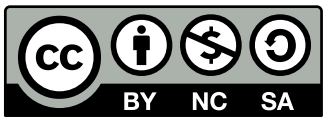
Aalto University
School of Science

Master's programme in Mathematics and Operations Research

Long-input summarization using Large Language Models

Emma Järvinen

Master's Thesis
2023



Aalto University
School of Science

Author Emma Järvinen

Title Long-input summarization using Large Language Models

Degree programme Mathematics and Operations Research

Major Systems and Operations Research

Supervisor Prof. Mikko Kurimo

Advisor M.Sc. (Tech) Viljami Raiski

Collaborative partner Smartbi Oy

Date 21 December 2023 **Number of pages** 77+5 **Language** English

Abstract

Large language models (LLMs) have shown remarkable capabilities in various natural language processing tasks. However, their output may not always meet the specific requirements or domain knowledge needed. The generated text may lack coherence or factuality, especially in summarization tasks with longer inputs. The increasing demand for automated summarization and the complexity of summarizing scientific content presents a unique challenge. This thesis will focus on the long-input summarization task of scientific articles using LLMs.

The thesis employs abstractive summarization techniques and explores two prominent strategies: fine-tuning LLMs and prompting. Fine-tuning involves adapting pre-trained models to the summarization task, leveraging their vast pre-trained knowledge, while prompting relies on structured instructions to guide LLMs in generating summaries without altering their weights. The research comprehensively analyzes these approaches, evaluating their strengths and weaknesses regarding summary quality, computational efficiency, and adaptability to the scientific domain.

Utilizing relatively small datasets from arXiv, the thesis showcases successful fine-tuning even with a limited amount of data, examines the impact of text preprocessing on fine-tuning, and optimizes prompt engineering through multiple prompts and a custom chunking algorithm. The limitations of numerical evaluation metrics in assessing text quality are critically analyzed. The research aims to provide valuable insights into long-input summarization, offering guidance on the effectiveness of fine-tuning and prompting strategies to enhance LLMs' capabilities in processing extensive and intricate textual documents.

The results of this study show that the fine-tuning strategy outperforms the prompting approach in the long-input summarization task of scientific articles. Furthermore, comparing non-fine-tuned and fine-tuned LLMs reveals that fine-tuning is a crucial step in using an LLM to summarize scientific articles, even with models pre-trained for summarization. We emphasize the limitations of using only numerical evaluation metrics in assessing the quality of generated texts and conclude that human evaluation is a vital part of ensuring the factuality and coherence of the generated summaries.

Keywords large language model, long-input summarization, fine-tuning, prompt engineering

Aalto-yliopisto
Perustieteiden
korkeakoulu

Tekijä Emma Järvinen

Työn nimi Pitkän tekstin tiivistäminen suurien kielimallien avulla

Koulutusohjelma Matematiikka ja operaatiotutkimus

Pääaine Systeemi- ja operaatiotutkimus

Työn valvoja Prof. Mikko Kurimo

Työn ohjaaja Diplomi-insinööri Viljami Raiski

Yhteistyötaho Smartbi Oy

Päivämäärä 21.12.2023 **Sivumäärä** 77+5 **Kieli** englanti

Tiivistelmä

Suuret kielimallit ovat osoittaneet merkittäviä kykyjä useissa luonnollisen kielen käsittelyä vaativissa tehtävissä. Niiden kyky tiivistää tekstiä on kuitenkin usein puutteellista. Kielimallien tiivistämä teksti ei välttämättä ole yhtenäistä tai todenperäistä, erityisesti tiivistettävän tekstin pituuden kasvaessa. Kasvava kysyntä automaattiselle tekstin tiivistämiselle yhdistettynä tieteellisen tekstin monimutkaisuuteen muodostaa ainutlaatuisen haasteen. Tämä diplomityö pyrkii kattavasti tarkastelemaan pitkien tieteellisten artikkeleiden tiivistämistä suurien kielimallien avulla.

Käytämme abstrakteja tekniikoita tieteellisten artikkeleiden tiivistämiseen. Hyödynnämme kahta eri strategiaa: kielimallien hienosäätöä sekä ohjeistamista. Hienosäätö sisältää esikoulutettujen mallien sovittamisen tiivistystehtävään hyödyntäen niiden laajaa ennalta opittua tietoa. Ohjeistaminen perustuu strukturoituihin ohjeisiin, jotka ohjaavat suuria kielimalleja tiivistelmien generoinnissa muuttamatta mallin painoja. Tutkimus analysoi perusteellisesti näitä lähestymistapoja arvioiden niiden vahvuuksia ja heikkouksia tiivistelmän laadun ja laskennallisen tehokkuuden osalta. Lisäksi arvioimme, kuinka nämä lähestymistavat toimivat tieteellisten tekstien kontekstissa.

Osoitamme, että suuria kielimalleja voi onnistuneesti hienosäätää hyödyntäen suhteellisen pientä tekstiaineistoa. Lisäksi tutkimme tekstin esikäsittelyn vaikutuksia hienosäätöön ja optimoimme ohjeistuksen suunnittelua käyttäen useita ohjeita ja räätälöityä paloittelualgoritmia. Tutkimus pyrkii tarjoamaan arvokkaita näkemyksiä pitkän tekstin tiivistämiseen suurien kielimallien avulla, tarjoten ohjeita mallien onnistuneeseen hienosäätöön sekä ohjeistamiseen.

Tämän työn tulokset osoittavat, että kielimallin hienosäätö tuottaa parempia tiivistelmiä tieteellisistä teksteistä verrattuna kielimallin ohjeistamiseen. Lisäksi osoitamme, että kielimallin hienosäätäminen on kriittistä, mikäli mallin halutaan tiivistävän tieteellistä tekstiä mahdollisimman hyvin, vaikka tehtävään käytetty kielimalli olisi esikoulutettu tekstin tiivistämiseen. Tutkimus osoittaa, että tekstin laadun arvioiminen ainoastaan numeerisia metriikoita käyttäen on puuttellista. Mikäli kielimallin tuottaman tiivistelmän faktat ja yhtenäisyys halutaan kattavasti varmistaa, on ihmisen hyödyntäminen arviointiprosessissa olennaista.

Avainsanat suuri kielimalli, pitkän tekstin tiivistäminen, kielimallin hienosäätö, kielimallin ohjeistaminen

Contents

Abbreviations

ANN	artificial neural network
BERT	bidirectional encoder representations from transformers
BLEU	bilingual evaluation understudy
C4	the colossal and cleaned version of common crawl
CNN	convolutional neural network
DAG	directed acyclic graph
FNN	feed-forward neural network
GELU	gaussian error linear unit
GSG	gap sentences generation
GSR	gap sentences ratio
ICL	in-context learning
LCS	longest common subsequence
LED	longformer-encoder-decoder
LLM	large language model
LSTM	long short-term memory
MLM	masked language model
MLP	multi-layer perceptron
NLP	natural language processing
ReLU	rectified linear unit
RL	reinforcement learning
RNN	recurrent neural network
ROUGE	recall-oriented understudy for gisting evaluation
SSL	self-supervised learning

1 Introduction

Large language models (LLMs), such as GPT-3, GPT-4 (Brown et al. [23]), and BERT (Devlin et al. [10]), have shown impressive capabilities in various natural language processing tasks aiming to generate coherent and contextually relevant text (Wang et al. [15]). However, their output may not always meet the specific requirements or domain knowledge needed in tailored real-life applications. Furthermore, the generated text may lack coherence or factuality, especially in summarization tasks with longer inputs (Koh et al. [16]). The demand for automated summarization has grown exponentially with the proliferation of digital content across various domains such as news, academia, legal documents, and social media (El-Kassas et al. [17]).

While the potential of developing an effective automatic text summarization system has attracted significant interest from the research community, automatic text summarization remains a challenging task. It is not applicable for wide practical use, particularly when it comes to summarizing longer texts such as scientific articles. Long-input summarization deals with the challenging task of summarizing extended documents or text passages, which often contain nuanced information and complex structures (Koh et al. [16]). In the context of machine learning, a document is considered long when current state-of-the-art models for a normal document cannot be implemented effectively due to hardware and model limitations. This work defines a long text as being longer than 1000 tokens.

This thesis will focus on the long-input summarization task of scientific articles using large language models. The abstractive summarization technique is used to summarize the articles. Abstractive summarization focuses on concisely summarizing a given text by interpreting and rephrasing the content rather than extracting and rearranging existing sentences or phrases (Kouris et al. [58]). We delve into two strategies for tackling the long-input summarization task - fine-tuning an LLM and employing prompting techniques. These two methodologies represent critical directions in natural language processing research, and their comparative analysis aims to contribute valuable insights into the best practices for addressing the long-input summarization task. We use relatively small datasets containing approximately 1k to 10k scientific articles. arXiv is used as a source for articles to ensure high-quality, human-generated text samples that align with the desired objectives and meet the requirements for the long-input summarization task, particularly in terms of text length.

Fine-tuning an LLM involves adapting a pre-trained language model to a specific task. For our summarization task, this means that we fine-tune a pre-trained model on a smaller, task-specific dataset consisting of text documents and their respective summaries. Fine-tuning allows the model to specialize in summarization by adjusting its weights, yet it requires a substantial amount of labeled data and domain-specific expertise. This approach has gained considerable attention and success in recent years, owing to its ability to leverage the immense pre-trained knowledge of LLMs and adapt it to the specifics of summarization (Zhang et al. [4], Phang et al. [7]).

On the other hand, prompt engineering relies on structured and domain-specific prompts to guide the LLM's behavior in generating a summary from a given text. Prompting is an efficient learning paradigm that allows LLMs to perform zero-shot

inference by conditioning on an instruction designed by humans (Zhou et al. [57]). A prompt is natural language text describing the task that an LLM should perform. These prompts are carefully designed to instruct the model to focus on critical elements, eliminate noise, and produce summaries from given texts. (Liu et al. [18]) In prompt engineering, a prompt containing the instructions for summarization and an article is fed to the model. An LLM aims to solve the given task at inference time without changing its weights. This approach is particularly appealing due to its flexibility and ease of use, as it doesn't require extensive fine-tuning with domain-specific data. However, it raises questions about the effectiveness of prompts, optimal prompt design, and potential limitations in handling longer documents.

We comprehensively examine these two approaches, fine-tuning and prompting, and evaluate their strengths and weaknesses in the context of long-input summarization. The evaluation will consider various aspects, including the quality of generated summaries, computational efficiency, models' adaptability to a scientific domain, and the need for labeled data. In fine-tuning, we leverage large pre-trained language models as a starting point and then fine-tune these models. In prompting, we do not fine-tune an LLM but rather guide the model via a prompt to summarize scientific articles. The performance of each model is evaluated using suitable evaluation metrics, and the generated summaries are compared against the reference texts. Technical accuracy of the generated texts is out of the scope of this work. We also analyze whether the existing metrics for evaluating text quality are sufficient.

This research aims to shed light on the path forward, addressing the challenges and opportunities of long-input summarization and contributing to the broader goal of making large language models more adept at processing lengthy and complex textual documents. The main contributions of this thesis can be summarized as follows:

- We comprehensively analyze the performance of various language models in the long-input summarization task of scientific articles. We utilize two datasets of scientific articles gathered from arXiv. The first dataset consists of approximately 9k articles from the reinforcement learning domain (RL dataset), whereas the other dataset contains approximately 1.5k articles from the domain of large language models (LLM dataset).

- We compare the fine-tuning and prompt engineering approaches that are used to adapt an LLM to a long-input summarization task. Additionally, we analyze how our fine-tuned models perform in the long-input summarization task of scientific articles by comparing them to previous work by others.

- We show that even with a limited amount of data, it is possible to fine-tune an LLM for a long-input summarization task successfully.

- We show that the fine-tuning strategy outperforms the prompting approach in the long-input summarization task of scientific articles. Furthermore, comparing non-fine-tuned and fine-tuned LLMs reveals that fine-tuning is a crucial step in using an LLM to summarize scientific articles, even with models pre-trained for summarization.

- We analyze how different text preprocessing techniques affect the fine-tuning of an LLM and present the preprocessing steps that lead to the best performance in the long-input summarization task.
- We analyze how prompt engineering can be optimized by formulating multiple prompts that guide the model in the summarization task and presenting a custom chunking algorithm.
- We emphasize the limitations of using only numerical evaluation metrics in assessing the quality of generated texts and conclude that human evaluation is a vital part of ensuring the factuality and coherence of the generated summaries.

1.1 Research questions

This thesis aims to answer the following research questions.

1. **How does a fine-tuned model perform in a long-input summarization task compared to the corresponding base model or prompt engineering?**

   Nowadays, it is possible to fine-tune large language models with custom data to generate text or apply prompt engineering to achieve the same goal. However, it is not always clear which of the abovementioned approaches works best in a summary generation task due to the many factors that must be considered. These factors include, among other things, the length of the input texts, the type of text, the availability of a suitable model for fine-tuning, data privacy, and financial and computational considerations. In this study, we comprehensively analyze the performance of language models in the long-input summarization tasks, leveraging fine-tuning and prompt engineering.

2. **How do varying hyperparameters and data preprocessing options impact the performance of the models and generated outputs?**

   It is crucial to preprocess the input text before feeding it to a model. Usually, tables, equations, and special characters are removed from the original text, and the text is lowercased (Tabassum et al. [51]). Preprocessing logic depends on the context and type of the original text. This thesis applies custom preprocessing logic to the input texts, and its effects are analyzed and compared to other preprocessing approaches.

   Prompt engineering provides a way to generate summaries without model fine-tuning. In prompt engineering, parameters such as chunk size, temperature, and choice of relatedness metric affect the model's outputs. These parameters should be chosen to achieve the highest quality of the generated texts. Additionally, crafting a suitable prompt that guides the model to summarize articles is significant. In this thesis, we study how prompt engineering can be utilized in the long-input summarization task. Furthermore, we aim to write an "optimal" prompt and study the effect of varying the chunk size.

Fine-tuning a pre-trained model requires selecting suitable values for the model's hyperparameters. How to define these hyperparameters should be carefully considered to make sure the model runs successfully within memory limits and time constraints. Furthermore, we must avoid overfitting the model and aim to achieve the highest performance possible regarding the evaluation metrics. Due to time and space complexities, we cannot test multiple hyperparameter sets in fine-tuning, even though it would have been interesting.

3. **How to comprehensively evaluate the quality of generated texts?**

   Generally, text can be evaluated using metrics such as ROUGE, BLEU, and BERTScore. However, these scores provide only a limited view of the quality of the text. (Gehrmann et al. [49]) If factors such as coherence and factuality are not considered, we might overestimate the quality of the generated text. For example, ROUGE scores can be reasonably high for a text that contains words similar to the original text, but its readability and factuality may be poor. This thesis analyzes the limitations of using only numerical metrics in assessing the quality of generated texts.

2 Background

This section aims to provide a thorough view of the large language models and text summarization. First, we will discuss artificial neural networks (ANNs), namely feed-forward and recurrent neural networks (FNN and RNN), that are predecessors of a Transformer model. Then, the Transformer model and its architecture are presented in detail, as each LLM used in this thesis relies on the Transformer architecture. Artificial neural networks are presented in Section 2.1, and its subsections cover FNN, RNN, and Transformer.

ANNs can be trained through various learning paradigms. In Section 2.2, we present three types of learning. Two common paradigms, supervised and unsupervised learning, are briefly discussed. However, Transformer-based models are usually trained with self-supervised learning; thus, we present this learning paradigm.

After covering ANNs, we will move on to discuss large language models, which are types of ANNs containing billions of parameters that they learn from massive amounts of textual data. We explain large language models in general in Section 2.3 and dive deeper into in-context learning and fine-tuning that adapt an LLM to a specific task in Section 2.4. In Section Evaluation metrics 2.5, we will introduce several evaluation metrics used to evaluate the quality of the generated content by an LLM. Furthermore, we cover text summarization, define relevant concepts, and discuss different techniques that can be used for summarizing text in Section 2.6.

2.1 Artificial neural networks

Artificial neural networks (ANNs, also shortened to NNs) are a class of machine learning models inspired by the structure and functioning of the human brain. They are a fundamental component of deep learning. This section will cover three types of ANNs: FNN in Section 2.1.1, RNN in Section 2.1.2, and Transformer in Section 2.1.3.

ANNs consist of interconnected nodes, often called neurons, organized into layers. Nodes are the basic computational units in a neural network. Each node receives one or more input signals, processes them, and produces an output. Nodes are usually organized into layers, including an input layer, one or more hidden layers, and an output layer. Weights represent connections between neurons. Each link has an associated weight that determines the strength of the connection. During training, these weights are adjusted to optimize the network's performance.

In addition, the activation function of a neuron defines the output of that neuron based on its input. It introduces non-linearities into the network, allowing it to learn complex relationships in the data. Standard activation functions include sigmoid, hyperbolic tangent (tanh), and rectified linear unit (ReLU) (Nair et al. [47]).

Neural networks are organized into layers. The input layer receives the initial data, the hidden layers process this data, and the output layer produces the network's final output. Deep neural networks have multiple hidden layers.

Neural networks learn from data by adjusting their weights during training. This process involves presenting the network with input-output pairs (training data),

computing the output, comparing it to the desired output, and adjusting the weights to reduce the error. (Wikipedia contributors [45])

2.1.1 Feed-forward neural network

A feed-forward neural network (FNN) is an artificial neural network in which the information flow direction is forward from the input nodes. FNNs have no cycles or loops. Modern feed-forward networks are trained using the backpropagation method. (Wikipedia contributors [33])

Let us next define equations for a hidden state and an output variable. Consider an input $\mathbf{X} \in \mathbb{R}^{n \times d}$, where n is the number of samples, and d is the number of inputs of each sample. The corresponding hidden state is defined as $\mathbf{H} \in \mathbb{R}^{n \times h}$, where h is the number of hidden units. Furthermore, we define a weight matrix $\mathbf{W}_{xh} \in \mathbb{R}^{d \times h}$, and an activation function ϕ. A logistic sigmoid or a hyperbolic tangent (tanh). We define a hidden state

$$\mathbf{H} = \phi_h(\mathbf{X}\mathbf{W}_{xh} + \mathbf{b}_h), \tag{1}$$

where $\mathbf{b}_h \in \mathbb{R}^{1 \times h}$ is a bias parameter. The output variable can be defined as

$$\mathbf{O} = \phi_o(\mathbf{H}\mathbf{W}_{ho} + \mathbf{b}_o). \tag{2}$$

One of the simplest FNNs is a single-layer perceptron. Perceptron uses a linear activation function. In perceptron, input values enter the layer and are multiplied by their corresponding weights. Each multiplication is added to get a weighted sum of all input values. If the sum of the values is above a specific threshold, usually set at zero, the value produced is often 1, whereas if the sum falls below the threshold, the output value is -1. The single-layer perceptron is often used in classification tasks (Wikipedia contributors [33]).

An extension of the perceptron is a multi-layer perceptron (MLP). The MLP consists of an input layer, one or more hidden layers, and an output layer. MLPs are fully connected, meaning that each node in one layer connects with a certain weight to every node in the following layer. Learning in MLP occurs by changing connection weights after each piece of data is processed based on the amount of error in the output compared to the expected result. This is an example of supervised learning and is carried out through backpropagation. Backpropagation performs a backward pass to adjust a neural network model's parameters, aiming to minimize the mean squared error. (Wikipedia contributors [34])

2.1.2 Recurrent neural network

Recurrent Neural Network (RNN) is a type of neural network architecture mainly used to detect patterns in a data sequence such as text or numerical time series and handwriting (Chung et al. [63]). RNNs are used widely for tasks such as handwriting recognition, machine translation, and time series prediction (Wikipedia contributors [21]). The main difference between FNN and RNN is that recurrent neural networks have cycles, and information is transmitted back into itself. RNN can be thought of as

a directed acyclic graph (DAG); that is, it consists of vertices and edges, with each edge directed from one vertex to another, such that following those directions will never form a closed loop (Wikipedia contributors [30]).

RNN has at least one hidden layer, and the information flows through the hidden layer(s). Next, we describe a process of passing information from the previous iteration to the hidden layer using the mathematical notation proposed by Zhang et al. [64]. Consider an input $\mathbf{X}_t \in \mathbb{R}^{n \times d}$ at timestep t, where n is the number of samples, and d is the number of inputs of each sample. The corresponding hidden state at timestep t is defined as $\mathbf{H}_t \in \mathbb{R}^{n \times h}$, where h is the number of hidden units. Furthermore, we define a weight matrix $\mathbf{W}_{xh} \in \mathbb{R}^{d \times h}$, and a hidden-state-to-hidden-state matrix $\mathbf{W}_{hh} \in \mathbb{R}^{h \times h}$. Lastly, an activation function ϕ, usually a logistic sigmoid or tanh function, is presented. The activation function is used to prepare the gradients for usage in backpropagation. Now that we have all the components to determine a hidden state, we get the following equation.

$$\mathbf{H}_t = \phi_h(\mathbf{X}_t \mathbf{W}_{xh} + \mathbf{H}_{t-1} \mathbf{W}_{hh} + \mathbf{b}_h), \tag{3}$$

where $\mathbf{b}_h \in \mathbb{R}^{1 \times h}$ is a bias parameter. The output variable can be defined as

$$\mathbf{O}_t = \phi_o(\mathbf{H}_t \mathbf{W}_{ho} + \mathbf{b}_o). \tag{4}$$

To emphasize the differences between FNNs and RNNs, the basic structure of both neural networks can be seen in Figure 1. The feed-forward neural network is on the left side, whereas the recurrent neural network is on the right.

A long short-term memory (LSTM) network is a popular type of RNN aimed to solve the vanishing gradient problem common when dealing with traditional RNNs (Wikipedia contributors [31]). The architecture in LSTM differs from the conventional RNN architecture. In LSTM, an additional module that learns when to remember and when to forget relevant information is added (Gers et al. [32]). For instance, in the context of NLP, the network can learn dependencies between types of words. "Long" in LSTM refers to the short-term memory that can last thousands of timesteps in RNN. Even though LSTM networks partially solve the vanishing gradient problem, they can still suffer from the exploding gradient problem.

A typical LSTM unit consists of a cell and three gates: an input gate, an output gate, and a forget gate. The gates regulate the flow into and out of the cell, which remembers the information received through the gates over an arbitrary time interval. The input gate regulates the inward flow of new information to the cell in its current state. On the other hand, the output gate is used to decide which information to output from the current state by considering both the current and the previous states. The forget gate controls what information is discarded from the previous state. Input and forget gates assign a value between 0 and 1 to the previous state, which decides whether to keep or discard the information. Output gates assign a value between 0 and 1 to the information using the previous and current states. A rounded value of 1 means to keep the information, and a value of 0 means to discard it. (Wikipedia contributors [31])

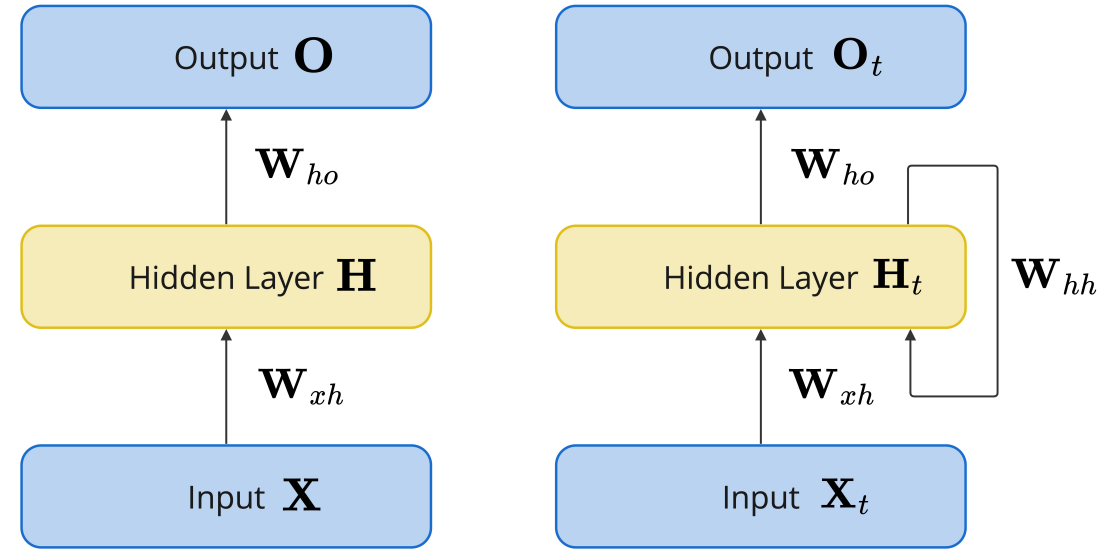


Figure 1: Simplified structure of FNN and RNN.

2.1.3 Transformer

This section provides a thorough walk-through of a Transformer model introduced by Vaswani et al. in the paper "Attention Is All You Need" [2] in 2017. Each model used in this thesis relies on the Transformer architecture. Thus, we must understand how these kind of models work. The Transformer model entirely relies on self-attention to compute representations of its input and output without using sequence-aligned RNNs or convolution.

The basic structure of a Transformer model is seen in Figure 2. The encoder maps an input sequence of symbol representations $(x_1, ..., x_n)$ to a series of continuous representations $z = (z_1, ..., z_n)$, from which the decoder then generates an output sequence $(y_1, ..., y_m)$ of symbols one element at a time. The model takes advantage of the previously generated output and the previous hidden state of the decoder when generating the following output. Before the processing in the encoder and decoder starts, input and output embeddings are fed to the positional encoding, which adds significant information about the order of the input sequence into each word.

The encoder and decoder components consist of layers called stacks. Let us denote the number of layers as N. Each encoder layer in a stack consists of two sublayers: a self-attention layer and a feed-forward neural network layer. Encoder layers are identical in structure but have unique weights. An encoder layer is described in more detail in Figure 3. A decoder layer has the corresponding sublayers, but in addition to the self-attention and the feed-forward neural network, there is an encoder-decoder

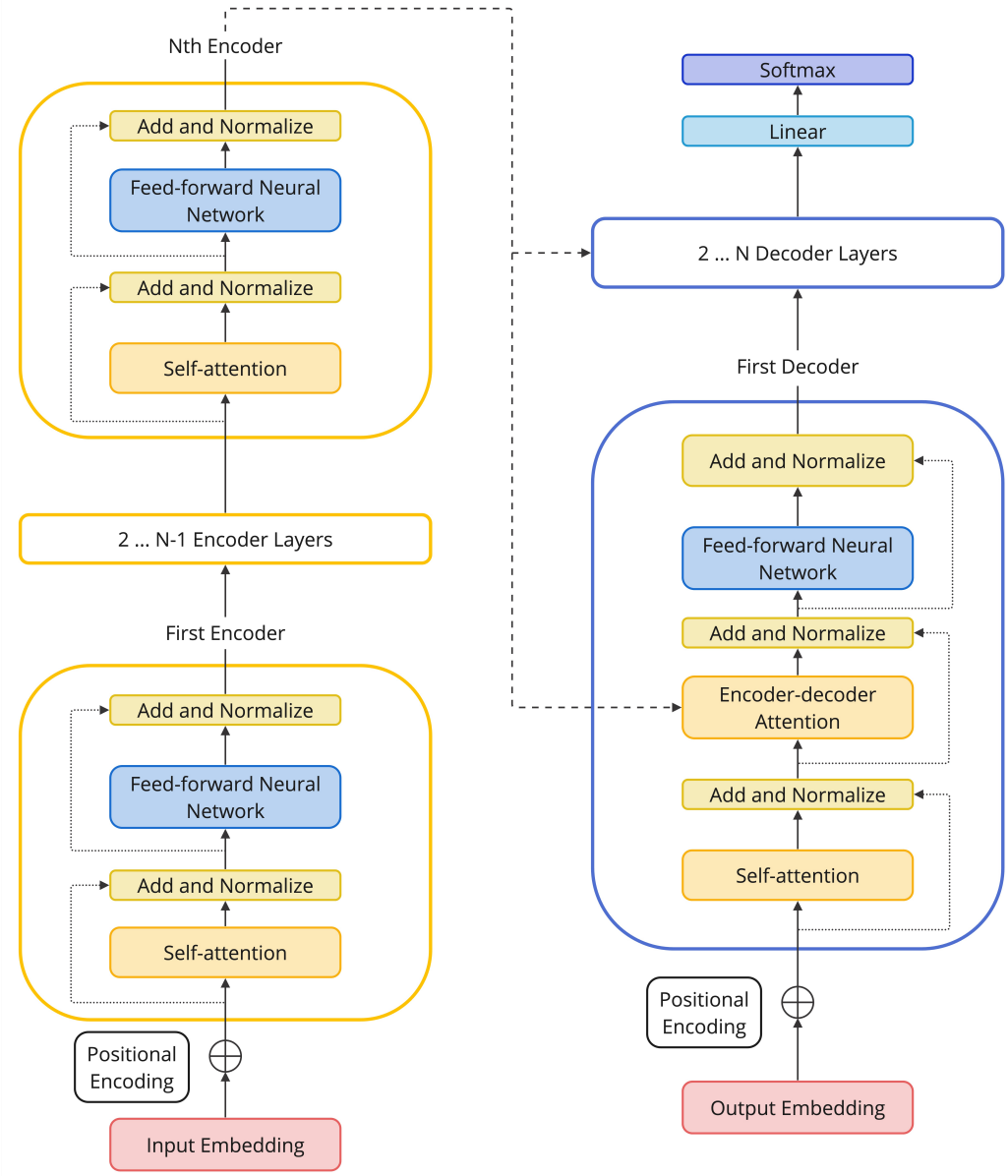


Figure 2: Basic structure of a Transformer model.

attention sublayer between the components, as mentioned earlier.

In the encoder part, the input embedding, or after the first layer, the output of the previous encoder layer, is fed to the self-attention layer. Let us consider the self-attention process in more detail. The self-attention process can be seen in Figure 4. We calculate query, key, and value vectors for every row (word) in X, q_j, k_j, and v_j, respectively. Here, $j \in \{1 \dots n\}$, where n is the number of input words. The corresponding matrices, that are seen in Equation 5, Q_i, K_i, and V_i are calculated by multiplying X with head-specific weights W_i^Q, W_i^K, and W_i^V. Here, $i \in \{1 \dots h\}$, where h is the number of heads or parallel layers.

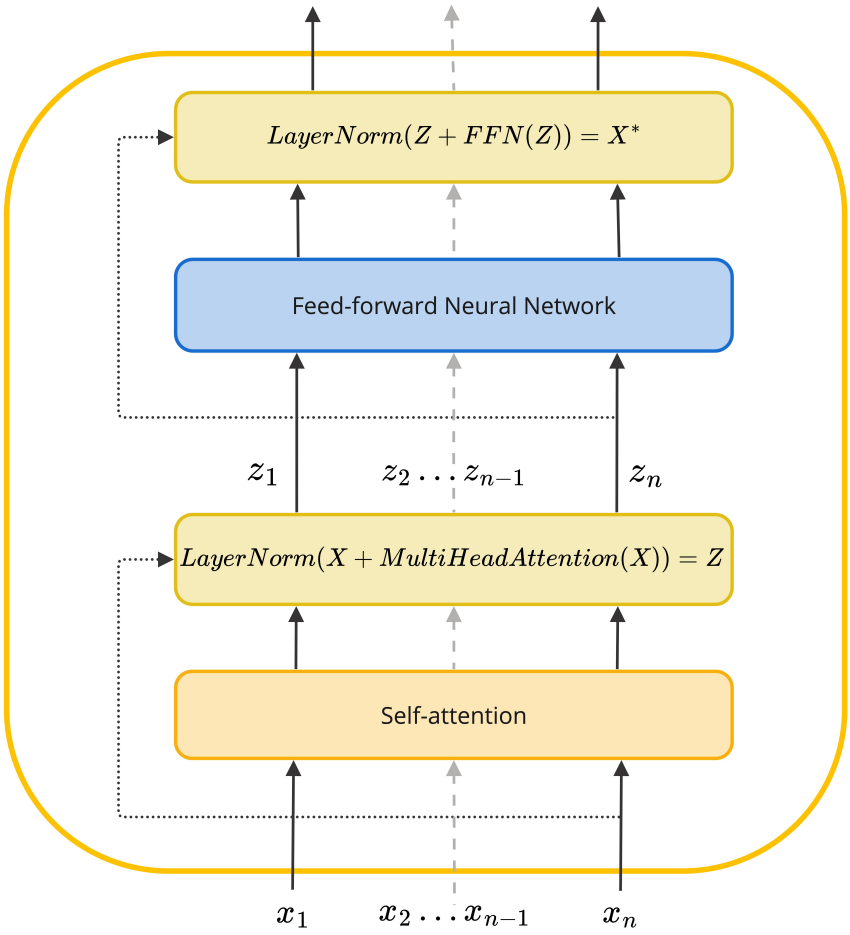


Figure 3: Detailed description of an encoder layer.

$$Q_i = XW_i^Q \tag{5a}$$
$$K_i = XW_i^K \tag{5b}$$
$$V_i = XW_i^V. \tag{5c}$$

Self-attention for a certain head$_i$, $i \in \{1 \ldots h\}$ is calculated as a function of Q_i, K_i, and V_i. It is defined as the inner product of Q_i and K_i divided by the square root of its dimensions. Each row in Q_i corresponds to an input word vector that is multiplied by every key vector in K_i. The process for calculating attention for an example query is visualized in Figure 5, and the corresponding equations are seen in 6.

$$\text{head}_i = \text{SelfAttention}(QW_i^Q, KW_i^K, VW_i^V) \tag{6a}$$

$$\text{SelfAttention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V. \tag{6b}$$

Calculating scores for a query means that a dot-product is taken with every key in K_i. A score can be understood as a factor used to determine the importance of other words in a sequence as we encode a word at a particular position. The scores are then scaled to have more stable gradients and passed into a softmax operation that normalizes the scores. The intuition behind the softmax operation is that we determine what words are expressed at a certain position. Then, each value vector is multiplied by the corresponding softmax score to keep the values of the words we want to focus on at a certain position intact. Lastly, we sum up the weighted value vectors to get the output vector of the self-attention layer at a certain position.

As can be seen from Figure 4, self-attention is calculated h times in parallel. Each head_i in multi-head attention is a unique representation of the input words multiplied by the corresponding weight matrices and thus expands the model's ability to focus on different aspects. The resulting h heads are concatenated into a single matrix multiplied with another weight matrix $(W^O)^T$ as seen in Equation 7.

$$\text{MultiHeadAttention}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O. \tag{7}$$

The above multiplication results in a matrix Z that contains information from all h attention heads. Before proceeding to the second sublayer, layer normalization is applied to the sum of the input word matrix X and the attention matrix Z. Layer normalization reduces training time and prevents the weight from exploding by restricting it to a certain range. This step can be seen in Figure 3. The resulting matrix Z can then be fed to the feed-forward neural network of the following structure, seen in Equation 8. This regular feed-forward network is applied separately to every attention vector z_i in Z. The FFN is unique in each sublayer. The FFN is used so that the output can be consumed by the next encoder block or, after the final encoder layer, the decoder block. One should note that layer normalization is applied to the sum of the input and output of the sublayers in each encoder block to employ a residual connection.

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2. \tag{8}$$

Now that we have covered the encoder part of the Transformer, seen on the left-hand side of Figure 2, we may proceed to the decoder side. The decoder is somewhat similar to the encoder part, the main difference being that an encoder-decoder attention sublayer exists between the already familiar components, the self-attention and FFN layers, from the encoder part. This encoder-decoder layer performs the already seen multi-head attention over the output of the encoder stack. The encoder-decoder layer takes the embeddings from both the input sequence and the target sequence to determine how each word in the input is related to the words in the target sequence. More specifically, in the encoder-decoder attention layer, the query matrix for the

target sequence is received from the self-attention layer below the current layer. In contrast, the key and value matrices are received from the output of the encoder stack.

The self-attention layer differs from the one seen on each encoder layer. It can only attend to earlier positions in the output sequence while generating the attention vectors. This ensures that predictions for a certain position depend only on the known outputs at positions before the current position. The masking of the future positions is done before the softmax step in the self-attention layer by setting them to $-\infty$. The query vectors (i.e., the collection of the attention vectors) describe how much each word is related to other words in the target sequence. As with the encoder, the residual connections are applied around each sub-layer, followed by layer normalization.

The decoder outputs vectors of floats that each correspond to a word. Now, these vectors are turned into words using linear and softmax layers. The linear layer is a fully connected neural network that projects a vector into a logit vector, where each float represents a score for a word. The size of the logit vector is the number of unique words in the model's output vocabulary. Finally, we may apply the softmax to the logit vector and obtain the corresponding probabilities. The highest probability is chosen, and the word associated with it is produced as an output of the Transformer at a current time step.

Now, we have covered the "basic" transformer architecture in detail. Later in this thesis, we will delve deeper into a few transformer-based language models that utilize the basic architecture with modifications.

2.2 Artificial neural network learning paradigms

Artificial neural networks can be trained using various learning paradigms, including supervised, unsupervised, and self-supervised learning. The choice of the learning paradigm depends on the nature of the task and the available data. Supervised learning is suitable for tasks with labeled data, unsupervised learning is suitable for tasks without labeled data, and self-supervised learning is suitable for creating supervision signals from the data itself. LLMs are usually trained through self-supervised learning; thus, we mainly focus on this learning paradigm in this section. Section 2.2.1 briefly covers supervised and unsupervised learning. In Section 2.2.2, we discuss self-supervised learning.

2.2.1 Supervised and unsupervised learning

In supervised learning, the model receives a set of labeled examples as training data and makes predictions for all unseen points. The model's job is to find a mapping between the given inputs and the corresponding labels such that the model generalizes to new data. Common use cases for supervised learning include classification, regression, and ranking problems. (Mohri et al. [59])

On the other hand, in unsupervised learning, the model is given unlabeled training data, and the model's job is to make predictions for all unseen points. It can be challenging to evaluate the performance of a model quantitatively since, usually,

no labeled example is available in that setting. Examples of unsupervised learning problems are clustering and dimensionality reduction. (Mohri et al. [59])

2.2.2 Self-supervised learning

Self-supervised learning (SSL) is a machine learning paradigm in which a model is trained on a dataset without explicit labels. Instead, the model leverages the inherent structure or information within the input data to generate its own supervision signals. Self-supervised learning incorporates ideas from both supervised and unsupervised learning. It is similar to unsupervised learning in that it learns from unlabeled data. Still, it is also supervised since the model creates its own pseudo-labels to learn from during training. The term "self-supervised learning" was first introduced in robotics, where training data is automatically labeled based on the relations between input sensor signals. The machine learning community further developed the idea. (Liu et al. [60])

SSL is particularly beneficial when labeled datasets are scarce or expensive to obtain. In self-supervised learning, the model typically formulates tasks that can be solved using the available input data without relying on external annotations. These tasks often involve generating missing parts of the input, predicting relationships between different parts, or other context-based predictions. Through this process, the model learns to extract meaningful features and representations from the data, which can later be fine-tuned for specific supervised tasks. (Qiu et al. [61])

One common example of self-supervised learning is pretraining a neural network on a massive amount of unlabeled text data using tasks like language modeling, where the model predicts the next word in a sequence. Once pre-trained, the model can be fine-tuned on a smaller labeled dataset for tasks like text classification or sentiment analysis. This demonstrates the transferability and effectiveness of features learned through self-supervised learning. Another example of a self-supervised task is the masked language model (MLM), which attempts to predict the masked words in a sentence given the rest of the words. (Qiu et al. [61])

2.3 Large language models

Large Language Models (LLMs) have garnered considerable attention recently due to their remarkable capabilities in understanding and generating human-like text. In this chapter, we delve into the evolving landscape of LLMs, their current applications, their promises for the future, and the history of their development. Additionally, we will explore their vulnerabilities and ethical implications while considering the potential use cases for fine-tuning these models and employing prompting techniques, especially in the context of long-input summarization.

2.3.1 Overview

Large language models are a type of language model notable for their ability to achieve general-purpose language understanding and generation. Large language

models are trained mostly through self-supervised learning on vast amounts of text from the internet, books, articles, and more. This text corpus gives them a broad understanding of human language and various topics. These models are designed to be general-purpose, meaning they can perform various language-related tasks, including text generation, translation, summarization, and question-answering. They don't need task-specific programming; they learn from the data they are trained on. The term "large" in large language models refers to the size of their neural network and the amount of training data. These models have millions or even billions of parameters, which allows them to capture intricate language patterns and nuances. Large language models are artificial neural networks, mainly Transformers (Vaswani et al. [2]). They are pre-trained using self-supervised learning (Wikipedia contributors [28]) and weak learning (Wikipedia contributors [29]). (Wikipedia contributors [20])

Large language models did not emerge overnight but resulted from decades of research and development in Natural Language Processing (NLP). The concept of using neural networks for NLP tasks dates back to the 1960s (Brill et al. [19]). However, the breakthroughs leading to LLMs can be traced to the advancements in deep learning, specifically in the form of deep neural networks known as RNNs and FNNs. The pivotal moment in the development of LLMs was the introduction of the Transformer architecture in the paper "Attention is All You Need" by Vaswani et al. in 2017 [2]. The Transformer architecture revolutionized NLP by introducing the concept of self-attention mechanisms, which enabled models to capture long-range dependencies in text efficiently. This breakthrough laid the foundation for the subsequent development of large-scale language models.

Large language models have since evolved and grown in size, with models like GPT-3 and GPT-4 (Brown et al. [23]) and BERT (Devlin et al. [10]) pushing the boundaries of what is possible in NLP. These models have billions of parameters and can comprehend and generate coherent text, enabling them to automate various language-related tasks. They have been employed in machine translation, sentiment analysis, chatbots, and content generation, significantly streamlining processes in many industries.

The future potential of LLMs is vast and exciting. As these models continue to evolve and grow in sophistication, they will likely play a pivotal role in various domains. In particular, long-text summarization is a promising application, providing companies with new ways to leverage their industry-specific documents. Fine-tuning or prompt engineering LLMs to generate concise and accurate summaries of complex papers could significantly enhance accessibility to specific knowledge.

While LLMs offer great promise, they are not without their vulnerabilities and ethical concerns. The potential for biases in generated content, the misuse of the technology for malicious purposes, and the environmental impact of training and deploying these massive models are areas of concern (Wu et al. [11]). Researchers and practitioners must address these issues responsibly and transparently.

Companies and organizations increasingly recognize the value of LLMs in automating tasks, enhancing customer interactions, and generating content (Daugherty et al. [62]). Fine-tuning or prompt engineering LLMs with domain-specific data can be a strategic move to harness the full potential of these models. For example, in

scientific research, companies can use fine-tuned LLMs to automatically summarize the latest developments in their respective fields, aiding decision-making processes and staying competitive.

Large language models have transformed the landscape of natural language understanding, offering unprecedented capabilities and opportunities for automation and innovation. However, their growth is accompanied by ethical considerations (Bender et al. [46]), and it is crucial to use these models responsibly. As we continue to explore the possibilities of LLMs, their ability to enhance accessibility to scientific knowledge through summarization holds immense promise and potential benefits for society.

2.3.2 Hallucination

Hallucination is a known problem in the context of large language models. In this context, the term "hallucination" refers to the generation of content that is not grounded in factual information or is not a faithful representation of reality. It occurs when the model produces output that appears coherent and contextually relevant but lacks accuracy or is entirely fabricated. (Ji et al. [48]) Hallucinations in language models can arise for various reasons, such as lack of fact-checking, over-optimization of training data, or ambiguity in training data.

Language models do not inherently possess a mechanism for fact-checking, so they may generate responses that sound plausible but are not necessarily true. Thus, a lack of fact-checking may occur. On the other hand, over-optimization of training data can happen if a model is overfitting its training data. In that case, it may generate content that aligns more closely with the training examples but does not necessarily reflect real-world accuracy. Furthermore, if the training data contains ambiguous or contradictory information, the model might "hallucinate" by filling in the gaps with its own interpretation.

Addressing hallucination in large language models is an ongoing research challenge. Researchers are working on developing methods to improve the factual accuracy and reliability of model outputs, but achieving perfection in this regard is a complex task. Users are encouraged to critically evaluate information generated by language models and verify it against reliable sources when necessary.

2.4 Customization of LLM

This section describes two methods, fine-tuning and in-context learning, that can be used to customize a large language model for a specific task. Fine-tuning and in-context learning are methods used to adapt pre-trained language models to specific tasks or contexts. Fine-tuning involves retraining the model on task-specific data, while in-context learning leverages the manipulation of input prompts to influence the model's output. Both approaches enable the customization of LLMs for diverse applications. First, we consider in-context learning in Section 2.4.1, then fine-tuning in Section 2.4.2.

2.4.1 In-context learning

In-context learning (ICL) is a task adaptation strategy that does not update the weights of the pre-trained large language model (Brown et al. [23]). In ICL, the model is conditioned on natural language instruction and a few demonstrations of the task and is then expected to complete further instances of the task simply by predicting what comes next. Recently, it has been argued by Awadalla et al. [27] that ICL leads to better out-of-domain performance when compared to fine-tuning, but there are references such as Mosbach et al. [26] that state the opposite. Next, we briefly discuss different approaches to learning within the context, namely zero-shot and few-shot learning.

Zero-shot learning represents a leap forward in LLMs' capabilities, allowing them to generate text without specific training or prior knowledge. As the name zero-shot suggests, in zero-shot learning, only instructions without any examples are provided in natural language to the model at inference (Brown et al. [23]). Zero-shot learning leverages the inherent capabilities of LLMs, which have been pre-trained on vast amounts of text from the internet. Zero-shot learning eliminates the need for domain-specific training. Instead, the model is guided by a simple prompt or instruction, typically in the form of a question or directive. Despite having no prior knowledge of the specific content, the model uses its general language understanding and reasoning abilities to generate a coherent and contextually relevant text.

However, the quality and relevance of the generated text can vary based on the prompt and the model's interpretation. It should be ensured that the generated content remains unbiased and factually accurate, and users must carefully craft prompts to achieve the desired results. Additionally, zero-shot summarization may struggle with highly technical or specialized content, and it should be carefully considered whether fine-tuning LLM could provide a better alternative for specific tasks or domains.

On the other hand, few-shot refers to the setting where the model is given a few demonstrations of the task at inference time as conditioning, but the model's weights are not updated (Brown et al. [23]). Few-shot learning with large language models involves training these models using a limited number of examples for specific tasks. Instead of extensive training data, LLMs rely on their pre-trained language understanding and patterns extracted from the provided examples to quickly adapt and generalize. In addition to few-shot learning, some literature distinguishes one-shot learning as its own. One-shot learning is the same as few-shot except that only one example is allowed at inference time, in addition to a natural language description of the task.

The critical advantage of few-shot learning is the ability of LLMs to learn and generalize from a small number of examples, leading to a significant reduction in the need for task-specific data. Their strong foundation in language prediction enables them to adapt to new inputs based on the patterns identified in the examples. However, ensuring the quality and relevance of the examples is crucial, as inaccuracies or inadequate representation can result in undesired outcomes. In text summarization, few-shot learning with relevant examples has significantly improved LLMs' accuracy (Brown et al. [23]). Training the model with curated prompt examples can incorporate

targeted language and capture essential topics in the generated summaries. This approach eliminates the need for rigid instructions, reducing the risk of overfitting the model to specific patterns.

In this thesis, we will make use of prompt engineering which is a zero-shot learning strategy. We aim to write an effective prompt and feed scientific articles to the model individually. The prompt asks the model to summarize the contents of an article. This summary can then be compared with the corresponding abstract from the original paper.

2.4.2 Fine-tuning

Fine-tuning has emerged as a popular approach in recent years within the domain of natural language processing and machine learning (Yu et al. [36]). This technique involves meticulously adjusting the model's weights, building upon a pre-trained architecture by exposing it to a carefully curated dataset tailored to the specific task. In most cases, these datasets encompass an array of thousands to hundreds of thousands of labeled examples, each serving as the fuel for the refinement process.

Fine-tuning should be considered when adapting to a new domain or genre (Mosbach et al. [26]). Fine-tuning necessitates the creation of a fresh, often large-scale dataset for each novel task, which makes it a resource-intensive endeavor. This demand for substantial data can pose a logistical barrier, particularly for tasks with limited pre-existing labeled examples. Moreover, fine-tuned models may exhibit a proclivity for overfitting the specific characteristics of the training data (Brown et al. [23]). This overfitting phenomenon can result in poor generalization when the model faces data distributions that differ from those encountered during training. In other words, the model might excel on the dataset used for fine-tuning but falter when confronted with real-world, out-of-distribution data. However, there exist results that state otherwise; Mosbach et al. [26] show that fine-tuned language models can, in fact, generalize well out-of-domain.

Another challenge is the possibility of fine-tuned models inadvertently seizing upon irrelevant or spurious features in the training data (Brown et al. [23]). Such an outcome can lead to a lack of robustness and fairness issues, as the model's performance may be inflated by its ability to exploit idiosyncratic artifacts within the training dataset.

Fine-tuning performs better with more high-quality examples. To fine-tune a model that performs better than using a high-quality prompt with the base model, at least a few hundred high-quality examples should be collected. Doubling the number of examples tends to increase the model's performance linearly. Increasing the number of examples is usually the best and most reliable way of improving performance. [1]

In this thesis, we fine-tune four large language models to adapt to a new domain, a dataset consisting of scientific articles that our model will then summarize using the corresponding abstract as a reference. The first dataset deals with reinforcement learning, and the other with large language models. These datasets and the fine-tuning process will be defined in more detail later.

2.5 Evaluation metrics

In this section, we present metrics that are commonly used to evaluate text generation models. Then, we consider the limitations of the evaluation metrics in Section 2.5.7.

The metrics we will discuss in this section are F1-score, ROUGE, BLEU, BERTScore, NIST, and cosine similarity. These metrics are used in NLP to compare generative or extractive tasks wherein two texts are to be compared. Machine-generated texts are typically assessed against a target text, representing the ideal output expected from the model. The 'generated text' refers to what the machine produces, while the 'target' or 'reference text' is the source we compare it to.

2.5.1 F1-score

F1-score is generally defined as,

$$F_1 = \frac{(\beta^2 + 1) \cdot \text{precision} \cdot \text{recall}}{\beta^2 \cdot \text{precision} + \text{recall}}, \tag{9}$$

where β is a term used to choose whether precision is favored over recall. Usually, β is set to 1, meaning the F1-score is evenly balanced. In this thesis, we use the evenly balanced version of F1-score, thus

$$F_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}. \tag{10}$$

Precision and recall are defined as

$$\text{precision} = \frac{\text{tp}}{\text{tp} + \text{fp}}, \tag{11}$$

$$\text{recall} = \frac{\text{tp}}{\text{tp} + \text{fn}}. \tag{12}$$

In Equations 11-12, tp stands for true positive, fp for false positive, and fn for false negative. Recall, as seen in Equation 12, is a function of its correctly classified examples (true positives) and its misclassified examples (false negatives), whereas precision, in Equation 11, is a function of true positives and examples misclassified as positives (false positives). [37]

Let's consider the F1 score in terms of the extractive summarization task. Precision is the number of correct words in the generated text divided by the total number of words in the generated text. Recall is the number of correctly predicted words divided by the total number of words in the target text. We may emphasize that the F1 score is computed based on correct words, not tokens. To illustrate the computation of the F1 score, we can consider the F1 score for the prediction *"Large"* when the reference answer is *"Large Language Models"* would be, $2 \cdot \frac{\frac{1}{1} \cdot \frac{1}{3}}{(\frac{1}{1} + \frac{1}{3})} = \frac{1}{2}$.

2.5.2 ROUGE

ROUGE stands for Recall-Oriented Understudy for Gisting Evaluation. ROUGE is a set of evaluation metrics commonly used to assess the quality of automatic or machine-generated summaries in natural language processing (NLP) tasks. It measures the similarity between a generated summary and one or more reference summaries. (Lin et al. [38])

ROUGE computes various metrics based on the overlap of n-grams (contiguous sequences of words) between the generated summary and the reference summaries. ROUGE-N, -L, and -S are the most commonly used ROUGE metrics.

ROUGE-N is an n-gram recall between a candidate summary and a set of reference summaries. ROUGE-N is computed as follows:

$$\text{ROUGE-N} = \frac{\sum_{S \in \{RefSum\}} \sum_{n\text{-gram} \in S} \text{Count}_{\text{Match}}(n\text{-gram})}{\sum_{S \in \{RefSum\}} \sum_{n\text{-gram} \in S} \text{Count}(n\text{-gram})}, \tag{13}$$

where n refers to the length of the n-gram, and $\text{Count}_{\text{Match}}(n\text{-gram})$ is the maximum number of n-grams co-occurring in a candidate summary and a set of reference summaries $\{RefSum\}$. ROUGE-N measures the overlap of n-grams between the generated and reference summaries. ROUGE-1 corresponds to unigrams, ROUGE-2 to bigrams, and so on.

On the other hand, ROUGE-L measures the longest common subsequence (LCS) between the generated and reference summaries. It captures the longest sequence of words that appears in both summaries while allowing for some rearrangements. ROUGE-S measures the skip-bigram overlap between the generated and reference summaries. A skip-bigram is a pair of words that appear in the same order in both summaries but may have other words between them.

The ROUGE-Lsum is a variation of the ROUGE-L metric, employing a slightly different calculation approach. Instead of evaluating the summary as a whole, ROUGE-Lsum operates at the sentence level using the ROUGE-L calculation method. It then combines these sentence-level results to generate the final score. This metric is particularly suitable for tasks where extracting information at the sentence level is crucial, such as in extractive summarization. In contrast to ROUGE-L, which assesses the entire summary without considering newlines, ROUGE-Lsum divides the text into sentences based on newlines. It computes the LCS for each pair of sentences and calculates the average score across all sentences. In simpler terms, ROUGE-Lsum provides a more detailed analysis by focusing on individual sentences, potentially offering greater insight into certain scenarios.

These metrics provide a quantitative assessment of how well a generated summary captures the important content and structure of the reference summaries. Higher ROUGE scores indicate better performance. To evaluate the performance of a language model using ROUGE, we'll compare the generated summary against one or more reference summaries and compute the ROUGE scores.

ROUGE is widely used in research and benchmarking of summarization systems and has become a standard evaluation metric in the NLP community. However, it's

important to note that ROUGE has limitations and may not capture all aspects of summary quality, such as coherence, fluency, and semantic understanding. Therefore, it should be used with other evaluation methods to provide a more comprehensive assessment of language model performance. (Gehrmann et al. [49])

2.5.3 BLEU

BLEU (Bilingual Evaluation Understudy) is a commonly used evaluation metric in natural language processing, particularly in machine translation tasks. It measures the similarity between a machine-generated translation and one or more reference translations. BLEU was originally designed to evaluate machine translation systems but has also been adapted for other NLP tasks. (Papineni et al. [39])

The BLEU score is based on precision, which measures the percentage of words or n-grams in the generated text that appear in the reference texts. BLEU computes a modified form of precision, called the brevity penalty, to account for differences in the lengths of the generated and reference texts.

We need one or more reference texts that serve as the ground truth or gold standard to calculate the BLEU score. The generated summary is compared against the reference summaries to compute the BLEU score.

First, we compute the N-gram overlap by calculating the modified precision for each N-gram up to a specified maximum order (usually $N = 4$). In this case, the N-grams represent phrases or sequences of words in the text. Then, we calculate the geometric mean of the modified precision scores for all N-grams. This step captures the overall similarity between the generated and reference texts.

Next, we apply the brevity penalty (BP) to account for length differences between the generated and reference texts. This step penalizes shorter or longer summaries to ensure fairness in the evaluation. The final BLEU score is obtained, ranging from 0 to 1, where a higher score indicates better similarity between the generated summary and the reference summaries.

BLEU can be computed as follows:

$$\text{BLEU} = \text{BP} \cdot \exp\left(\sum_{n=1}^{N} w_n \log p_n\right), \tag{14}$$

where BP stands for brevity penalty calculated such that

$$\text{BP} = \begin{cases} 1 & c > r \\ e^{1-\frac{r}{c}} & c \leq r \end{cases}. \tag{15}$$

In Equation 14, we first compute the geometric mean of the modified n-gram precisions, p_n, using n-grams up to length N and positive weights w_n summing to one. Then, we multiply the result by an exponential brevity penalty factor seen in Equation 15, where c is the length of the candidate translation and r is the effective reference corpus length. (Papineni et al. [39])

By using BLEU in summary generation tasks, we can quantitatively evaluate the quality of generated summaries. However, one should note that BLEU has its

limitations. As stated in "BERTScore: Evaluating text generation with BERT" by Zhang et al. [40], BLEU provides a simple and general measure but fails to account for meaning-preserving lexical and compositional diversity.

2.5.4 BERTScore

BERTScore is a metric that evaluates the quality and similarity of text based on the principles of Transformer-based language models, particularly BERT (Bidirectional Encoder Representations from Transformers) (Devlin et al. [10]). This metric deviates from traditional evaluation measures by considering contextual information, capturing semantic nuances, and enabling a more comprehensive assessment of text similarity (Zhang et al. [40]).

The theoretical foundation of BERTScore lies in its use of token embeddings derived from pre-trained BERT models. These embeddings capture the contextual dependencies between words within sentences, enhancing the metric's ability to discern semantic relationships. This contextual awareness sets BERTScore apart from traditional metrics like BLEU and ROUGE.

Next, we define BERTScore according to the original paper by Zhang et al. [40]. Given a reference sentence $x = \langle x_1 \dots x_k \rangle$ and a candidate sentence $\hat{x} = \langle \hat{x}_1 \dots \hat{x}_l \rangle$, we use contextual embeddings to represent the tokens, and compute matching using cosine similarity, which is defined later in this section, optionally weighted with inverse document frequency scores. Importance weighting can be easily applied to BERTScore, but it is not discussed in detail here. These contextual embeddings can generate different vector representations for the same word in different sentences depending on the surrounding words, which form the context of the target word (Zhang et al. [40]).

The embedding model generates a sequence of vectors $\mathbf{x} = \langle \mathbf{x}_1 \dots \mathbf{x}_k \rangle$, and $\hat{\mathbf{x}} = \langle \hat{\mathbf{x}}_1 \dots \hat{\mathbf{x}}_l \rangle$ for both the tokenized reference and the candidate sentences, respectively. These vector representations are then compared using cosine similarity. Finally, given a reference sentence and a candidate sentence, precision and recall are combined to calculate the F1-score that is:

$$P_{\text{BERT}} = \frac{1}{|\hat{x}|} \sum_{\hat{x}_j \in \hat{x}} \max_{x_i \in x} \mathbf{x}_i^T \hat{\mathbf{x}}_j, \tag{16}$$

$$R_{\text{BERT}} = \frac{1}{|x|} \sum_{x_i \in x} \max_{\hat{x}_j \in \hat{x}} \mathbf{x}_i^T \hat{\mathbf{x}}_j, \tag{17}$$

$$F_{\text{BERT}} = 2 \cdot \frac{P_{\text{BERT}} \cdot R_{\text{BERT}}}{P_{\text{BERT}} + R_{\text{BERT}}}. \tag{18}$$

BERTScore is computationally efficient and scalable, making integration into various NLP applications practical. Pre-trained models and libraries are readily available, simplifying its adoption within research and industry settings. BERTScore aligns closely with human judgment, suggesting its effectiveness in assessing the coherence and fluency of generated text. However, BERTScore is not exempt from

limitations. It may face challenges in domains with specialized language or constrained computational resources.

2.5.5 NIST

NIST is based on the BLEU metric but with some alterations. NIST is mainly used in translation tasks but can be used in other NLP tasks. BLEU calculates n-gram precision, adding equal weight to each one, whereas NIST calculates how informative a particular n-gram is. NIST also differs from BLEU's calculation of the brevity penalty as small variations in translation length do not impact the overall score as much. The NIST metric was designed to improve BLEU by rewarding the translation of infrequently used words. This was intended to further prevent the inflation of SMT evaluation scores by focusing on common words and high-confidence translations. As a result, the NIST metric uses heavier weights for rarer words. The final NIST score is calculated using the arithmetic mean of the n-gram matches between SMT and reference translations. In addition, a smaller brevity penalty is used for smaller variations in phrase lengths. The reliability and quality of the NIST metric are shown to be superior to the BLEU metric in many cases. (Doddington et al. [41])

According to a definition in Doddington et al. [41], NIST is calculated as follows. First, information weights are computed using N-gram counts over the set of reference texts according to the following equation:

$$\text{Info}(w_1, \ldots, w_n) = \log_2 \left(\frac{\text{The \# of occurrences of } w_1, \ldots, w_{n-1}}{\text{The \# of occurrences of } w_1, \ldots, w_n} \right). \tag{19}$$

Then, using Equation 19, we compute NIST:

$$\begin{aligned} \text{NIST} = \sum_{n=1}^{N} \left\{ \frac{\sum_{\text{co-occurring } w_1,\ldots,w_n} \text{Info}(w_1, \ldots, w_n)}{\sum_{w_1,\ldots,w_n}(1)} \right\} \\ \cdot \exp\left\{ \beta \cdot \log^2 \left[\min\left(\frac{L_{sys}}{\bar{L}_{ref}}, 1 \right) \right] \right\}, \end{aligned} \tag{20}$$

where $N = 5$ is a constant, and β is chosen to make the brevity penalty factor 0.5 when the number of words in the system output is 2/3 of the average number of words in the reference translation. Furthermore, $\bar{L}_{ref}$ is the average number of words in a reference text, averaged over all reference texts, whereas L_{sys} is the number of words in the text being scored.

2.5.6 Cosine similarity

Cosine similarity can be used to measure the similarity between tokenized words or sentences. Cosine similarity is a similarity measure between two non-zero vectors defined in an inner product space. Cosine similarity is the dot product of the vectors

divided by the product of their lengths, i.e., the cosine of the angle between the vectors. It can be calculated such that

$$\text{cosine similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}, \tag{21}$$

where A_i and B_i are ith components of vectors $\mathbf{A}$ and $\mathbf{B}$, respectively. Cosine similarity does not depend on the magnitudes of the vectors but only on their angle. The cosine similarity is measured at $[-1, 1]$.

2.5.7 Limitations

There are many challenges with evaluating language models that produce natural language. The evaluation metrics presented above produce numeric values of the performance of a model that generates natural language. Most of these metrics measure only the similarity between model output and references but cannot estimate any quality aspects of the generated outputs. We may ask whether a single metric is adequate to describe the performance of a language model if it cannot measure, e.g., factuality or coherence. Furthermore, we haven't yet discussed human evaluation, which is a critical part of the evaluation of language models.

Human evaluation improves the evaluation process as humans can consider more nuances in generated language than automated metrics. However, great investments are required in using humans to evaluate the performance of language models. It is both expensive and time-consuming. Additionally, human evaluation can be challenging if the evaluators are not familiar enough with the material used to train and test a language model. Our task is to fine-tune a model or use prompt engineering techniques to summarize long scientific articles. To evaluate the generated summaries, the evaluator should be familiar with the original papers' topics to check the summary's factuality and overall quality. This would require the evaluator to read and internalize the original paper and its abstract. The evaluator could only critically assess the generated summary's overall quality by understanding the original paper.

In this thesis, we cannot use comprehensive human evaluation even though it would give us more insight into the performance of the models. Instead, we will assess only a few texts generated by the models and compare them to the corresponding reference texts.

Gehrmann et al. [49] state that most automated metrics measure only the similarity between model output and references and fail to consider quality aspects. On the other hand, human evaluations have a high variance and rarely produce replicable results due to insufficient documentation. Recognizing these limitations, recent academic papers have critiqued existing evaluation methods or proposed novel approaches. However, there is a notable gap between these recommendations and actual implementation. The disconnect arises from a misalignment of incentives, as there is little motivation for researchers to invest in rigorous evaluations when the focus is primarily on publishing new models or modeling techniques. While adopting general evaluation techniques could facilitate the integration of evaluation advancements into model development,

their creation demands valuable resources, such as model outputs for validation and test sets or substantial amounts of human assessments. Challenges, such as refining datasets, involve collaborative and iterative processes among multiple researchers. Consequently, a cyclic dependence emerges, where enhancements in the evaluation of generation models are contingent on using improved evaluations by generation models themselves. (Gehrmann et al. [49])

We can conclude that a single metric cannot describe a language model's performance. Despite the known problems, many publications still use only one metric, e.g., ROUGE, to demonstrate improvements over prior systems. This is also the case with the language models we will utilize in this thesis. The only evaluation metric used when assessing the performance of the models was ROUGE. These models will be described later.

2.6 Text summarization

As early as the late 1950s and early 1960s, it was suggested that text summarization by computer was feasible but not straightforward. Since then, the interest in text summarization has grown remarkably as the amount of digital content has grown exponentially (El-Kassas et al. [17]). In the following subsections, we consider key concepts related to the text summarization task using large language models. First, we focus on extractive and abstractive summarization in Section 2.6.1 and then consider a long-input summarization task in Section 2.6.2.

Let us first give a definition and requirements for a summary. A summary is a text that is produced from one or more texts, that contains a significant portion of the information in the original text(s), and that is no longer than half of the original text(s) (Hovy et al. [24]). A summary must fulfill the following requirements: it must be shorter than the original input text, contain the vital information (the user defines importance) of the original, and not other, totally new information (Hovy et al. [24]).

Indicative summaries provide an idea of what the text is about without giving any content, whereas informative ones provide some shortened version of the content. On the other hand, extracts are summaries created by reusing portions (words, sentences, etc.) of the input text verbatim, while abstracts are created by re-generating the extracted content. [24]

2.6.1 Extractive and abstractive summarization

In extractive summarization, the focus is on selecting relevant sentences from the input text to construct a summary. It is to decide whether each sentence should be included in the summary. Thus, this approach treats the task as a binary classification problem. The famous architecture for extractive summarization is BERT, which serves as a baseline and can be further fine-tuned for specific domains. Determining which sentences matter in a summary is subjective and depends on the field, leading to variations in extractive summarization approaches. As an example of extractive summarization, SBERT (Reimers et al. [25]), a widely used library, facilitates the creation of summaries with the flexibility to specify the desired number of sentences.

SBERT is a modification of the pre-trained BERT network that uses siamese and triplet network structures to derive semantically meaningful sentence embeddings that can be compared using cosine similarity (Reimers et al. [25]).

On the other hand, abstractive summarization aims to generate a concise summary by comprehensively considering all the information in the input text. Unlike extractive summarization, it does not rely on using exact sentences from the original text but instead focuses on capturing the essence of the content through paraphrasing. (Kouris et al. [58]) Typical architectures for abstractive summarization follow a Transformer-based approach, similar to extractive summarization, but can be additionally fine-tuned for specific domains such as books, research papers, interviews, and legal documents. These models typically support larger input sizes. Given the extensive data required to fine-tune Transformer-based architectures for new use cases, most summarizers are built as abstractive summarizers, leveraging the abundantly available datasets. The models used in this thesis use abstractive summarization to generate text.

2.6.2 Long-input summarization

Long-input summarization deals with the challenging task of summarizing lengthy documents or text passages, which often contain nuanced information and complex structures. Long-input summarization is particularly vital in the context of scientific literature, legal documents, news articles, and other information-rich domains. Recently, Transformer-based LLMs have emerged as promising tools for long-input summarization, offering transformative capabilities in condensing lengthy texts into concise summaries. Significant research efforts have been made to improve automatic text summarization systems, and numerous studies on the challenges of extending these systems to the long document domain have emerged (Koh et al. [16]).

To further discuss long-input summarization, we must define what is considered "long" input. The literature lacks an agreement on the specific word limit distinguishing long and short inputs in a summarization task. However, Koh et al. [16] differentiates short and long document summarization based on document length, content breadth, and coherence level. In the context of machine learning, a document is deemed long when current state-of-the-art models face challenges in effective implementation due to hardware and model constraints. Koh et al. propose that a benchmark dataset with an average source document length surpassing 3000 tokens qualifies as a "long document" since many state-of-the-art summarization systems are restricted to 512 to 1024 tokens. This study defines a long input as exceeding 1000 tokens, while an input with 1000 tokens or fewer is considered short. The 1000-token limit corresponds to an average of 750 words (OpenAI [53]).

In contrast to short documents, long documents are typically structured into sections to aid user comprehension, each containing distinct content related to the document. Summarizing long documents poses a challenge for LLMs as they grapple with fluency, redundancy, and semantic coherence issues when integrating key information from various sections into a cohesive summary. As document length increases, the volume of informative, non-redundant content tends to expand. However, summary length is constrained by user perceptions of reasonableness. Empirical evidence reveals

that the relative length of summaries compared to source documents significantly decreases with longer documents, resulting in the loss of non-essential information in long document summarization. Diverse user preferences and expectations further complicate this task (Koh et al. [16]).

One of the main challenges in long-input summarization with Transformer-based models is a quadratic scaling of memory consumption as the input length doubles (Beltagy et al. [3]). As an example, let us consider a Transformer-based summarization model with 12 encoder and 12 decoder layers, which were initially pre-trained on short input sequences of 512 tokens and then fine-tuned for specific tasks that involved longer input sequences of 16 384 tokens, with an output length of 512 tokens in both cases.

Contrary to the usual practice where fine-tuning is less resource-intensive than pre-training, in this case, fine-tuning is more resource-intensive and slower. This is primarily due to the significant increase in input sequence length during fine-tuning, which results in a quadratic scaling of memory consumption for self-attention operations in the encoder. Specifically, the encoder's self-attention consumes 1024 times more memory during fine-tuning than pretraining due to the 32-fold increase in input length.

Even if an efficient Transformer variant is used, such as Longformer (Beltagy et al. [3]), to mitigate memory and computation issues, both the encoder's self-attention and the decoder's cross-attention operations still consume 32 times more memory compared to the pretraining stage. Additionally, operations such as the Feed-Forward Network (FFN) that scale linearly with input length also significantly increase the computational requirements during training and inference. (Phang et al. [7])

The increasing reliance on LLMs for summarization also raises quality and ethical concerns. In addition to the computational challenges in long-input summarization, another major challenge is the lack of comprehensive evaluation metrics that could be used to estimate the quality of generated texts. Even though metrics such as ROUGE and BLEU exist, which are used to measure the similarity between the original and generated text, human evaluators are needed to assess coherence and factuality. Ensuring that generated summaries are unbiased, accurate, and free from unintended errors remains challenging. Researchers and practitioners must exercise caution and develop robust evaluation methodologies to maintain the integrity of summarization outputs.

LLMs are continuously developed to address these challenges. The goal is to create models that understand language nuances, have contextual awareness, and have extensive pre-trained knowledge that enables them to generate summaries that capture the essence of lengthy texts while maintaining readability and coherence. However, computational resources and time complexity must be considered while developing LLMs that can process long-input sequences. Furthermore, these models should be able to produce more human-like summaries in their structure and fluency.

The applications of long-input summarization with LLMs are manifold. In academia, researchers could use fine-tuned LLMs to generate concise summaries of research papers, facilitating rapid literature review and knowledge dissemination. Legal professionals can expedite document review processes by automating the extraction

of critical information from legal documents. In journalism, LLMs can assist in summarizing lengthy investigative reports, making news more accessible to readers.

Long-input summarization with LLMs represents a significant challenge in NLP. These models empower us to distill vast volumes of information into concise, informative summaries, offering potential across various domains. As LLMs evolve and improve, their role in long-input summarization is expected to expand, enhancing our ability to navigate the ever-expanding sea of textual information efficiently and precisely.

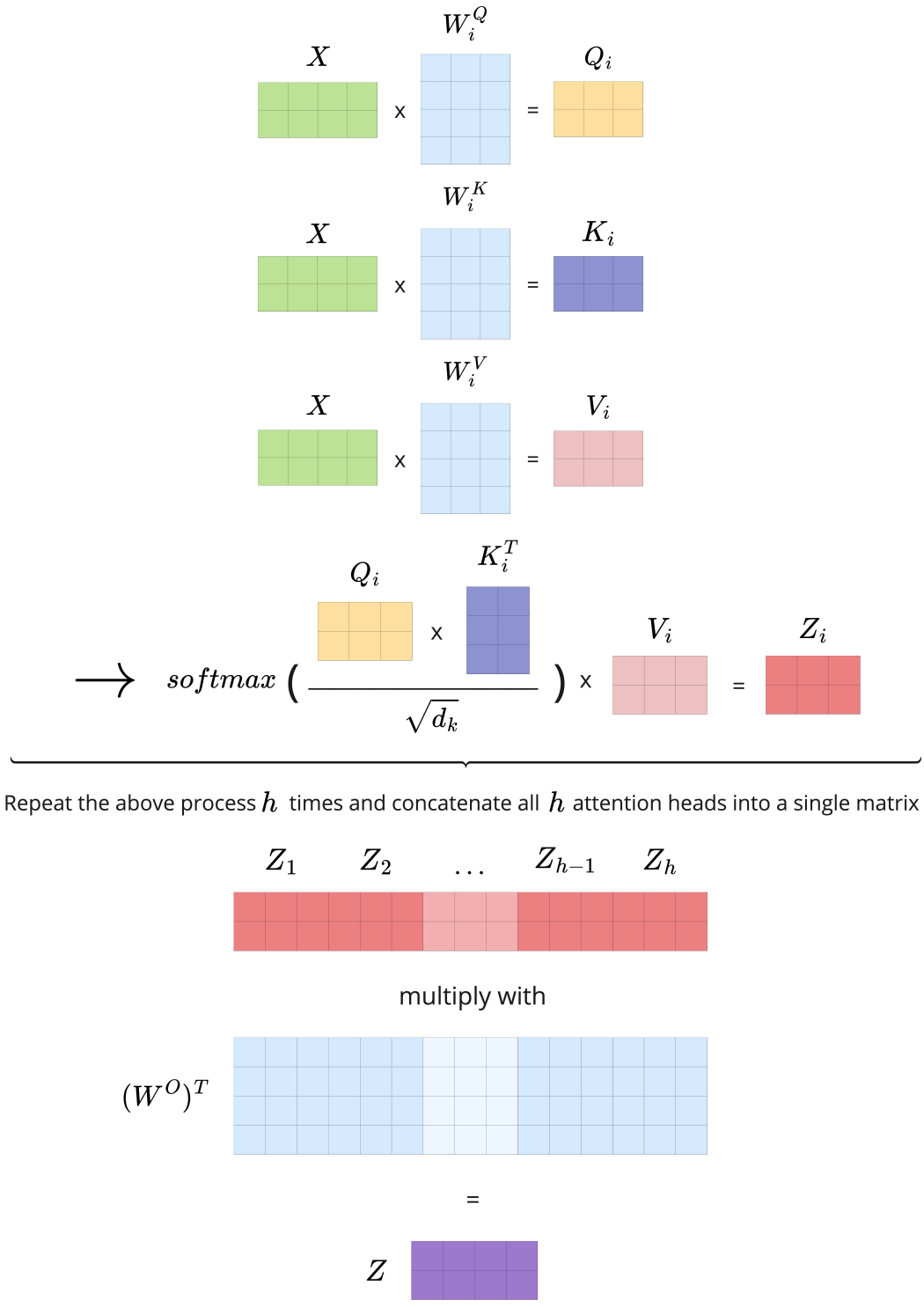


Figure 4: Self-attention process in detail.

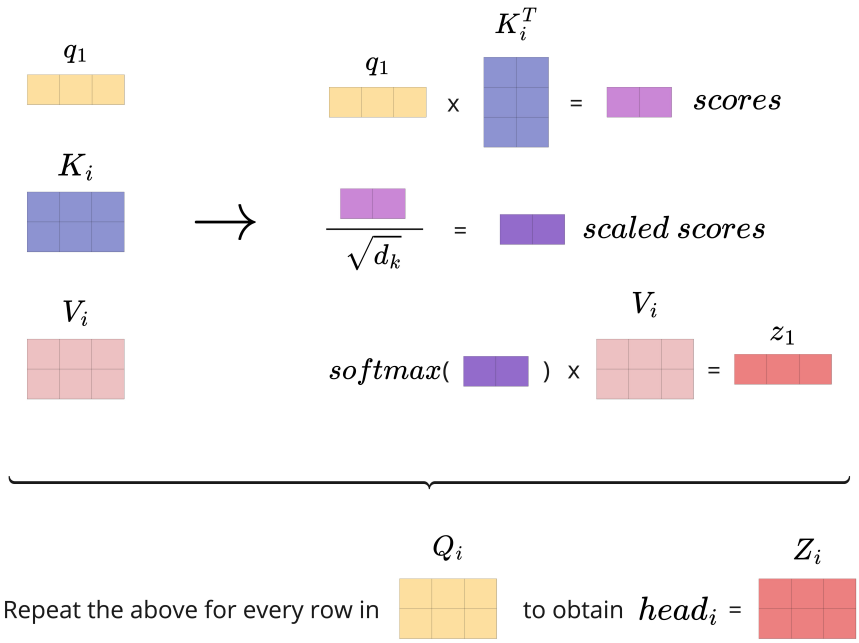


Figure 5: Attention for the first query (i.e. row) in Q_i.

3 Research material and methods

In this section, we present the data and methods used in the summarization task, where our objective is to generate an abstract from a scientific article. First, we describe our data and explain the corresponding preprocessing logic in Sections 3.1 and 3.2. As for the methods, fine-tuning and prompt engineering are applied to LLMs to improve the base models' ability to generate relevant text. The models we fine-tune or query by prompt engineering are presented in Section Models in our summarization task 3.3. We will explain the fine-tuning and prompt engineering processes in Sections 3.4 and 3.5.

3.1 Data

This section describes the data used in the summarization task. The objective was to find at least one dataset containing lengthy texts with the same main topic, a similar vocabulary, and a similar text structure. The dataset should be large enough to learn necessary and significant information during fine-tuning. However, it should not be too large as we have limited computational resources in fine-tuning models. Another objective was to compose the dataset such that the main topic of the texts wouldn't be too broad. Otherwise, the fine-tuning may not capture all the necessary information from the data provided to the model. To meet these requirements, scientific articles were chosen as a type of text. The objectives were also chosen to mimic real-world cases where gathering a dataset containing over 10 thousand or even smaller amounts of text documents is often impossible.

Two datasets were used throughout fine-tuning different LLMs and in in-context learning with GPT-3.5. Both datasets contain scientific articles from arXiv [35]. The first dataset consists of roughly 1000 articles. Three datasets were concatenated into one more extensive set to get enough data for training, validation, and testing. The datasets were gathered via arXiv API using the following keywords: "large language model", "transformer model", and "neural network". The number of articles is also limited to 1000, 100, and 500 in each set, respectively. The other dataset was fetched through arXiv API using the "reinforcement learning" keyword. The size of this dataset is approximately 9000 articles. Later in this thesis, we will refer to these datasets as LLM and RL datasets.

The vocabulary in the scientific articles is domain-specific, thus providing an exciting factor to consider in the long-input summarization task we'll perform with our datasets. The size of both datasets is relatively small as we aim to mimic real-world situations where it is not often plausible to have tens of thousands of domain-specific articles with their corresponding summarizations.

LLM and RL datasets were preprocessed before feeding the texts to the LLMs. Custom preprocessing logic was created to process the articles into a suitable form. These articles contain many mathematical notations, equations, tables, and figures. Even though it would be interesting to be able to consider all the information that can be gained from these parts of the article, it is out of the scope of this thesis. The preprocessing is discussed in the next section.

After preprocessing, data was split into training, testing, and validation sets. The dataset sizes are seen in Table 1. The average, minimum, maximum, and median word counts before tokenization and after preprocessing for the articles and summaries in LLM and RL datasets are seen in Table 2. The word count was calculated by splitting the text by space.

Table 1: Sizes of train, test, and validation sets for LLM and RL datasets.

Dataset	Train	Test	Validation	Total
LLM	1165	146	146	1457
RL	7350	919	919	9188

Table 2: The average, minimum, maximum, and median word counts before tokenization and after preprocessing for the inputs (articles) and labels (summaries).

Dataset	Column	Average	Median	Minimum	Maximum
LLM	Input	5456	4993	246	54894
LLM	Label	153	151	32	274
RL	Input	5776	5239	44	58832
RL	Label	155	153	14	297

3.2 Preprocessing

We start this section by briefly discussing common practices in preprocessing textual data for language models. Then, we will present the preprocessing logic applied to the data before feeding it to the models.

First, we define what text preprocessing means in the context of NLP. We use the definition by Tabassum et al. [51], which states that preprocessing a text means bringing the document to a format that is easily understandable, predictable, and analyzable by the machine through the various machine learning algorithms. According to Tabassum et al. [51], some of the widely used pre-processing techniques are:

1. **Sentence segmentation** Sentence segmentation refers to breaking a text document or corpus into individual sentences. This alleviates the problem of identifying word boundaries so that further processing can be done sentence-wise. Usually, the text is segmented into sentences when a full stop occurs.

2. **Change to lower case** Capital letters usually occur at the beginning of a sentence, in proper nouns and abbreviations. It is argued that making all words lowercase is one of the most effective and simplest steps in text preprocessing. Lowercasing text is especially effective when the dataset is significantly sparse. The main benefit of lowercasing is that it decreases the number of word vectors formed

from each word. For example, the words "Dog" and "dog" would produce different word embeddings without lowercasing, even though the words have exactly the same meaning.

3. **Tokenization** In tokenization, we split sentences into tokens that can be words, characters, and punctuations. Usually, the splitting criteria in tokenization is a space or a punctuation. Based on OpenAI's Token Calculator [53], a general rule of thumb is that 75 words approximately equals 100 tokens, and 1 token approximately equals 4 text characters.

4. **Parts-of-Speech tagging** In POS tagging, each word in a sentence is tagged with its corresponding grammar class, such as a noun, a pronoun, a verb, an adjective, etc. The sentences are tagged based on the sentence segmentation done before POS tagging. In natural language, some words can represent more than one part of speech at different times. POS tagging can be a challenging problem when word forms are ambiguous.

5. **Stop words removal** Stop words removal refers to removing words from a text that do not have significance in terms of a certain NLP task. Stop words usually contains words such as "the", "is", and "and". However, it is case dependent on what words should be included in the set of stop words. In classification problems, stop words are usually removed. For other NLP tasks, such as summarization, we omit this step as we would lose the meaning of the sentences if we were to remove the stop words.

6. **Removal of punctuations** Special characters such as exclamation marks, commas, or apostrophes are removed in this step. Removal of punctuation makes the text less noisy for the machine to read.

7. **Stemming** Stemming is a word-shortening technique that converts a word to its base root. In stemming, we shorten the suffixes in a text such that the semantic meaning of all different forms remains the same. For example, "reading" would change to "read" when stemmed. However, "riding" changes to "rid", which no longer makes sense. It depends on whether we should use stemming and what words should be shortened to their base root.

8. **Lemmatization** Lemmatization is similar to stemming in that both techniques bring a word to its base. Lemmatization differs from stemming in that the word is brought to its base, but the resulting word is meaningful, which is not necessarily the case in stemming. For example, "riding" would change to "ride".

Let us next describe our preprocessing logic. Initially, we have all articles in PDF format, fetched through arXiv API. Furthermore, we have an abstract for each article in CSV format.

1. **Lowercase the abstract and remove special characters** First, we lowercase the abstract, then remove the following special characters: [,ˆ, <,], +, ?, >, and numbers.

2. **Read the article as blocks with pymupdf library** `pymupdf` is a high-performance Python library for data extraction, analysis, conversion, and manipulation of PDF (and other) documents. From this library, we use a function that extracts a page's text blocks as a list of items. Blocks of text are easier to identify and modify for further purposes than plain text, as each block contains information about the block type that can be used to determine whether the block contains an image or text. We will remove blocks containing an image. Note that the remaining text blocks contain tables and equations in text format that are removed later in this process.

3. **Drop non-ASCII characters** From each text block, we remove non-ASCII characters. Scientific articles contain multiple equations and tables that often have non-ASCII characters. We cannot account for information from tables or equations, so all non-ASCII characters are removed.

4. **Remove abstract from the article** We must carefully remove the abstract, i.e., target, from each article to use the articles as inputs for the summarization task. We use sequence matching to find the block that contains the abstract by comparing each block's content to the abstract we already have in CSV format.

5. **Remove Acknowledgements, Appendices, and References from the article** Acknowledgments, appendices, and references are part of most articles in our datasets. However, these sections do not contain significant information regarding the abstract. Thus, these parts are removed from the original article.

6. **Remove short blocks** Due to unknown reasons, some blocks may contain only a few characters. These short blocks are removed to parse the article's text from PDF to CSV format as clearly as possible.

7. **Remove links** All links are removed from each article.

8. **Lowercase the article and remove special characters** First, we lowercase the article, then remove the following special characters: [, ^, <,], +, ?, >, and numbers.

In general, programming with PDF files is challenging due to the complexity of the format.

3.3 Models in our summarization task

This section thoroughly describes the models that are used in our summarization task. Each model is either fine-tuned or queried by prompt engineering. We consider five models: `T5` in Section 3.3.1, `PEGASUS` in Section 3.3.2, `PEGASUS-X` in Section 3.3.3, `LED` in Section 3.3.4, and `GPT-3` in Section 3.3.5. The models were selected as a result of comprehensive research and some requirements. Models are required to be able to handle long inputs as our data consists of lengthy scientific articles. Due to limited GPU resources in fine-tuning, we must also find models that don't exceed the available

memory during the fine-tuning process. If one exists, we also present previous results for a summarization task of scientific articles or similar texts in Section 3.3.6.

3.3.1 T5

A set of T5 -models (Text-to-Text Transfer Transformer) was first published and introduced by Raffel et al. in 2020 [5]. The base structure of T5 follows the Transformer model proposed by Vaswani et al. [2] apart from small deviations. These deviations include removing the Layer Norm bias, placing the layer normalization outside the residual path, and using a simplified form of position embeddings.

T5 comes in different sizes; small version with ~60M parameters, base model with ~220M parameters, large version with ~770M parameters, 3B and 11B versions. The small T5 -model has $L = 6$ layers for encoder and decoder, a hidden size of $H = 512$, a feed-forward layer of $F = 2048$ in size, and $A = 8$ self-attention heads. We will later fine-tune the small version of T5.

The basic idea in T5 is that it takes text as an input and produces new text as an output, making it a text-to-text model and allowing it to generalize to different types of text processing problems. The text-to-text framework allows the same model, objective, training procedure, and decoding process to be applied to every task considered.

T5 was pre-trained on the C4 (The Colossal and Cleaned version of Common Crawl) dataset. C4 contains text from 350M web pages; the dataset is 750 GB. A maximum sequence length of 512 was used in the pre-training. The hyperparameters of the pre-training $\texttt{T5}_{\texttt{SMALL}}$ are seen in Table 3. T5 was first pre-trained using a simple denoising objective. In a denoising objective, the model is trained to predict missing or otherwise corrupted tokens in the input. The objective randomly samples and drops out 15% of tokens in the input sequence. The C4 dataset contains unlabeled data; thus, the objective does not require labels but teaches the model generalizable knowledge. This "general knowledge" is later utilized in downstream tasks.

After pre-training, the model was fine-tuned on downstream tasks, including machine translation, question answering, abstractive summarization, and text classification. The model has been evaluated on various English-based NLP tasks, such as document summarization, translation, and question answering. Before training, a task-specific prefix was added to each input text to train T5 on various tasks. These prefixes are typically short, e.g., for a translation task from English to Finnish, the prefix is "translate English to Finnish: ". Similarly, the prefix for a summarization task is "TL;DR: ". Naturally, these prefixes are used in fine-tuning too. Since we are especially interested in the summarization task, we will later in Section Results on the summarization tasks 3.3.6 take a look into the performance of the T5 -model on the summarization task that was achieved by Raffel et al.

A non-anonymized version of the CNN/Daily Mail dataset (Hermann et al. [56]) by See et al. [8] was used to measure the downstream performance on the abstractive summarization task. The CNN/Daily Mail dataset is widely used in short summarization tasks. The dataset contains document-summary pairs, where the document represents a news article. (Koh et al. [16])

3.3.2 PEGASUS

PEGASUS (Pre-training with Extracted Gap-sentences for Abstractive Summarization) was implemented by Zhang et al. in 2020 [4]. PEGASUS belongs to the family of sequence-to-sequence models. The base architecture of PEGASUS is a standard Transformer encoder-decoder. In contrast to models such as T5 (Raffel et al. [5]) and BART (Lewis et al. [6]), in PEGASUS, important sentences are removed or masked from an input document and are generated together as one output sequence from the remaining sentences, similar to an extractive summary. As an outcome, two models were published; $\text{PEGASUS}_{\text{BASE}}$ with 223M parameters, and $\text{PEGASUS}_{\text{LARGE}}$ with 568M parameters. PEGASUS models are intended to be used for abstractive summarization. We will later fine-tune $\text{PEGASUS}_{\text{LARGE}}$.

The pre-training objectives of PEGASUS models were tailored for abstractive text summarization, i.e., generating a summary-like text from an input document. When developing PEGASUS, there were two main pre-training objectives: Gap Sentences Generation (GSG) and Masked Language Model (MLM). The idea behind GSG is that whole sentences from a document are masked, and these so-called gap sentences are then concatenated into a pseudo-summary. To approximate a summary even more closely, sentences that appear to be important/principal to the document are selected. The m sentences that are masked are selected using three strategies: based on their importance to the document, selecting m sentences at random, or selecting the first m sentences from a document. These strategies are called Principal, Random, and Lead, respectively. A gap-sentences ratio (GSR) is the percent of sentences masked from a document, and finding an ideal value for this hyperparameter in pre-training was crucial.

In the other objective, MLM, 15% of tokens in the input text were selected. These selected tokens are 80% of the time replaced by a mask token, 10% of the time replaced by a random token, and 10% of the time kept unchanged. Both GSG and MLM were used as objectives in training the Transformer encoder, but MLM was not included in the final $\text{PEGASUS}_{\text{LARGE}}$ model as it did not improve downstream tasks at a large number of pre-training steps. Additionally, it was found that the Principal strategy, i.e., selecting the most important sentences from a document, worked best in pre-training PEGASUS.

$\text{PEGASUS}_{\text{LARGE}}$ has $L = 16$ layers for encoder and decoder, a hidden size of $H = 1024$, a feed-forward layer of $F = 4096$ in size, and $A = 16$ self-attention heads. GSR of 30% was found to be an effective ratio in pre-training $\text{PEGASUS}_{\text{BASE}}$ as a result of ablation studies with $\text{PEGASUS}_{\text{BASE}}$, and testing the base model's performance and comparing the effect of having GSR from 15% to 75% on downstream datasets. However, when scaling up to $\text{PEGASUS}_{\text{LARGE}}$, GSR was increased to 45% to achieve a similar number of gaps as the base model.

Two large text corpora, C4 and HugeNews, were used as the pre-training corpus. HugeNews is a dataset containing 1.5B articles from news and news-like websites from 2013 to 2019. A maximum sequence length of 512 was used in the pre-training phase. The hyperparameters of the pre-training are seen in Table 3. PEGASUS was evaluated on 12 downstream summarization tasks spanning news, science, stories,

instructions, emails, patents, and legislative bills. Additionally, human evaluation studies were leveraged to validate the quality of the generated summaries. For downstream summarization, 12 different, publicly available abstractive summarization datasets were used to fine-tune both $\texttt{PEGASUS}_{\texttt{BASE}}$ and $\texttt{PEGASUS}_{\texttt{LARGE}}$.

One of the 12 datasets used for downstream summarization is arXiv's scientific articles (215k in size) published by Cohan et al. [54]. The arXiv dataset is a long scientific document summarization dataset collected from the arXiv.org scientific repository. This dataset represents the earliest work on large-scale, long document summarization datasets (Koh et al. [16]). As we will utilize two custom samples collected from arXiv when fine-tuning our models, this gives us another benchmark that we will use to compare our fine-tuned model's performance and the results of $\texttt{PEGASUS}_{\texttt{LARGE}}$ on the arXiv dataset. The arXiv dataset contains documents considerably longer than the maximum input length of 512 in pre-training. Because sinusoidal positional encodings [2] generalize well, $\texttt{PEGASUS}_{\texttt{LARGE}}$ was fine-tuned with 1024 tokens. However, the input length of 1024 tokens is not even close to the average input length of the articles in the arXiv dataset.

In the real world, gathering a large dataset (>100k) of supervised examples to fine-tune a summarization model is often difficult. Thus, Zhang et al. simulated low-resource summarization using small samples of the 12 datasets mentioned above and fine-tuning both $\texttt{PEGASUS}_{\texttt{BASE}}$ and $\texttt{PEGASUS}_{\texttt{LARGE}}$ with the truncated datasets. The sample sizes used to fine-tune the models were 10, 100, 1k, and 10k. The promising results indicate that it is possible to achieve high ROUGE scores even with a limited number of data.

3.3.3 PEGASUS-X

In 2022, PEGASUS-X was first introduced by Phang et al. in the paper "Investigating Efficiently Extending Transformers for long-input Summarization" [7] to address the ongoing challenges of long-input summarization. Compared to models like PEGASUS and T5 described above that are pre-trained with a maximum input sequence length of only 512, PEGASUS-X is an extended version of $\texttt{PEGASUS}_{\texttt{LARGE}}$, pre-trained with additional long-input sequences up to 16K tokens. PEGASUS-X is intended to be used for long-input summarization.

As mentioned in Section Long-input summarization 2.6.2, the main limitation of training LLMs with long-input sequences is the quadratic scaling of the memory requirements, as computing the attention mechanism in Transformer-base models is quadratically dependent on the input length. Phang et al. experimented with different Transformer architectures for pre-training and fine-tuning with long inputs. First, the memory consumption of the models with longer input sequences was investigated by swapping the encoder of PEGASUS for more efficient encoder architectures such as Big Bird (Zaheer et al. [12]) that use sliding window and global token attentions, and Performer (Choromanski et al. [13]) which factorizes attention matrices via orthogonal random features. In addition to Big Bird and Performer, two variants of local attention Transformer encoders, Local and Global-Local, are introduced. Local is a simple block-local Transformer, where encoder input tokens are divided into non-overlapping

blocks, which means that tokens can only attend to other tokens within the block. Global-Local is an extension of the local Transformer where a set of global tokens with learnable embeddings are added, and these tokens can attend to and be attended from every encoder token.

Different encoder architectures were tested on short and long (e.g., arXiv scientific articles by Cohan et al. [54]) summarization tasks. The full-attention Transformer performed best among the short summarization tasks, whereas Global-Local performed best among the long summarization tasks. Both Global-Local and Local encoders had a good balance between performance and efficiency, and thus, they were improved further. Results with encoder variants can be seen in the model paper by Phang et al. [7].

A series of approaches were tested to improve Global-Local and Local encoder architectures and later construct an efficient model `PEGASUS-X`. First, a staggering of local attention blocks was added. In block-local attention, the information is not shared across blocks; the same block boundaries are used across all layers. In staggering, the boundaries are shifted in every other layer, allowing cross-block interactions with minimal additional complexity or computational cost. In the Global-Local model, the following variant was considered. The global token representations were supplied to the decoder, and an additional encoder-decoder cross-attention that attends only to the global tokens before performing cross-attention over the encoded tokens was introduced. Both of these changes in Local and Global-Local models improved the performance significantly. This is particularly surprising as the Global-Local model already has a set of global tokens used in cross-block interactions.

The Global-Local model's performance on the summarization task was measured by varying the block size and the number of global tokens. The key takeaway from this experiment was that increasing either block size or global tokens up to a certain point leads to improved performance. Still, the increase is seen in computation time and memory consumption. A block size of 64 and 32 global tokens was used for the rest of the experiments with the Global-Local encoder.

Different position encoding schemes were tested on a full-attention Transformer, pre-trained with an input length of 512, and fine-tuned with an input length of 2048 for the long-input tasks. Each scheme was applied to the model's encoder and decoder parts. `PEGASUS` and Vaswani et al. [2] use a sinusoidal position encoding, but also the bucket-based relative position encoding scheme of `T5`, RoPE (Su et al. [14]), absolute position embeddings and no position encoding was tested. `T5` performed best based on ROUGE scores, but it was almost twice as slow as the other schemes. On the other hand, sinusoidal position encoding and RoPe performed only a little worse than `T5`, and thus, the sinusoidal position encoding was chosen to be used in `PEGASUS-X`.

Furthermore, Local and Global-Local models were tested on summarization tasks by scaling the encoder and decoder layers with a fixed total number of layers. Decoder-heavy models seemed to perform slightly better for Local models. In contrast, a balanced encoder-decoder outperforms both encoder- and decoder-heavy alternatives for Global-Local models, but the difference is not remarkable. In a long-input summarization task, the inputs are considerably longer than the resulting output, which leads to different computational trade-offs depending on the balance between

the number of encoder and decoder layers. Encoder-heavy models require more memory due to the long inputs, whereas decoder-heavy models are somewhat slower at inference.

Next, it was investigated whether it was better to pre-train a model with an efficient encoder from the beginning or use the efficient encoder in fine-tuning or additional pre-training directly after the model is first pre-trained with a full-attention Transformer on short input sequences. In both approaches, pre-training was performed with short input sequences of 512 tokens. Four pre-training and fine-tuning approaches were tested and evaluated using ROUGE scores on long summarization tasks with four different block sizes: Transformer in pre-training and Local in fine-tuning, Local in both pre-training and fine-tuning, Transformer in pre-training and Global-Local in fine-tuning, and Global-Local in both pre-training and fine-tuning. Block sizes used were 4, 16, 64 and 256. Out of all these 16 different approaches, Global-Local performed best in pre-training and fine-tuning with block sizes of 64 and 256.

A second-to-last experiment tested Global-Local and Local encoders with different pre-training schemes. In this experiment, short-input pre-training, with 512 input tokens and 256 output tokens, and long-input pre-training, with 4096 input tokens and 256 output tokens, was considered. In total, five different pre-training formats were tested for both Local and Global-Local encoders. The pre-training formats consisted of short-input pre-training for 100% of tokens with 500k and 1M steps, short-input pre-training for 75% of tokens, and then long-input pre-training for the remaining 25% of tokens with 1M steps, short-input for 50% of tokens and long-input for 50% of tokens with 1M steps, and lastly long-input for 100% of tokens with 1M steps. Considering the ROUGE scores of the ten approaches, the Global-Local encoder outperformed the Local encoder, and pre-training should have both short- and long-input sequences to achieve the best performance.

Lastly, an encoder-decoder cross-attention was investigated to reduce memory consumption. Using an efficient encoder already reduces the memory requirements as it scales linearly rather than quadratically in input length. However, the other major factor in memory consumption is the encoder-decoder cross-attention, as each decoder layer attends separately to the long encoder representations. To address this issue, a model with a Global-Local encoder was pre-trained and fine-tuned using cross-attention only in some of the decoder layers. The main outcome was that the best practice is to pre-train a Global-local model with full cross-attention and then drop cross-attention from a subset of the decoder layers in fine-tuning. This decreases the performance only a little but reduces memory consumption.

Based on the experiments, PEGASUS was adapted to handle long-input summarization. The Global-Local encoder architecture with block staggering was chosen. During pre-training, many global tokens and large block sizes were used. An additional pre-training was conducted using 4096 tokens for 300k steps. During fine-tuning, input sequences of up to 16384 tokens were used depending on the task.

As a result, two models were released: a smaller PEGASUS-X_{BASE} with 272M parameters, and PEGASUS-X with 568M parameters. Furthermore, Phang et al. evaluated a series of proposed efficient Transformer architectures and other model tweaks. They reported their efficacy and trade-offs on computational resources when

applied to long-input summarization tasks. The pre-training phase of PEGASUS-X followed a similar process as was done with PEGASUS in the corresponding model paper by Zhang et al. [4]. Additionally, pre-training with long-input sequences of 4096 tokens was performed with a reduced GSR of 5.625% (the original masking ratio of 45% was used in pre-training PEGASUS). The reduced GSR matched the 8x increase in input sequence length. $\text{PEGASUS-X}_{\text{BASE}}$ has $L = 16$ layers for encoder and decoder, a hidden size of $H = 1024$, a feed-forward layer of $F = 4096$ in size, and $A = 16$ self-attention heads. The hyperparameters of the pre-training are seen in Table 3.

Model	Corpus	LR	BS	n_s	$l^{\text{max}}_{\text{input}}$	$l^{\text{max}}_{\text{target}}$
T5_{SMALL}	C4	0.01*	128	2^{19}	512	-
$\text{PEGASUS}_{\text{LARGE}}$	C4/HugeNews	0.1	8192	500k	512	256
$\text{PEGASUS-X}_{\text{BASE}}$	C4	0.1	512	500k (300k)	512 (4096)	256

Table 3: Hyperparameters of the pre-training. *The learning rate for the first 10^4 steps is 0.01, then exponentially decays until pre-training is over. For $\text{PEGASUS-X}_{\text{BASE}}$, the numbers in brackets refer to an additional long-input pre-training.

3.3.4 LED

The Longformer-Encoder-Decoder (LED) was introduced in 2020 by Beltagy et al. in [3]. LED is a Longformer variant for supporting long document generative sequence-to-sequence tasks. Sequence-to-sequence refers to a process of transforming sequences from one domain to another. Specifically, LED has an attention mechanism that scales linearly with sequence length, whereas standard Transformer-based models have a self-attention mechanism that scales quadratically with the sequence length. Thus, with LED, it is relatively easy to process long documents. LED was pre-trained with the Books corpus and English Wikipedia. Furthermore, a subset of the Realnews dataset with documents longer than 1200 tokens and a subset of the Stories corpus were used in pre-training. LED is developed to be suitable for modeling sequence-to-sequence tasks with long documents.

Longformer's attention mechanism combines a windowed local-context self-attention and an end task-motivated global attention that encodes inductive bias about the task. Both attention types are needed but serve different purposes. Local attention is primarily used to build contextual representations, whereas, with global attention, Longformer can build full sequence representations for prediction.

Next, we present the design and implementation of Longformer's attention mechanism in detail. The standard full self-attention has the computational complexity of $O(n^2)$, where n is the length of the input sequence. Full attention is seen in Figure 6 a). To achieve linear scalability, the full self-attention matrix is sparsified according to an attention pattern that specifies pairs of input locations attending to one another. First, the attention pattern employs a fixed-size window of attention surrounding each

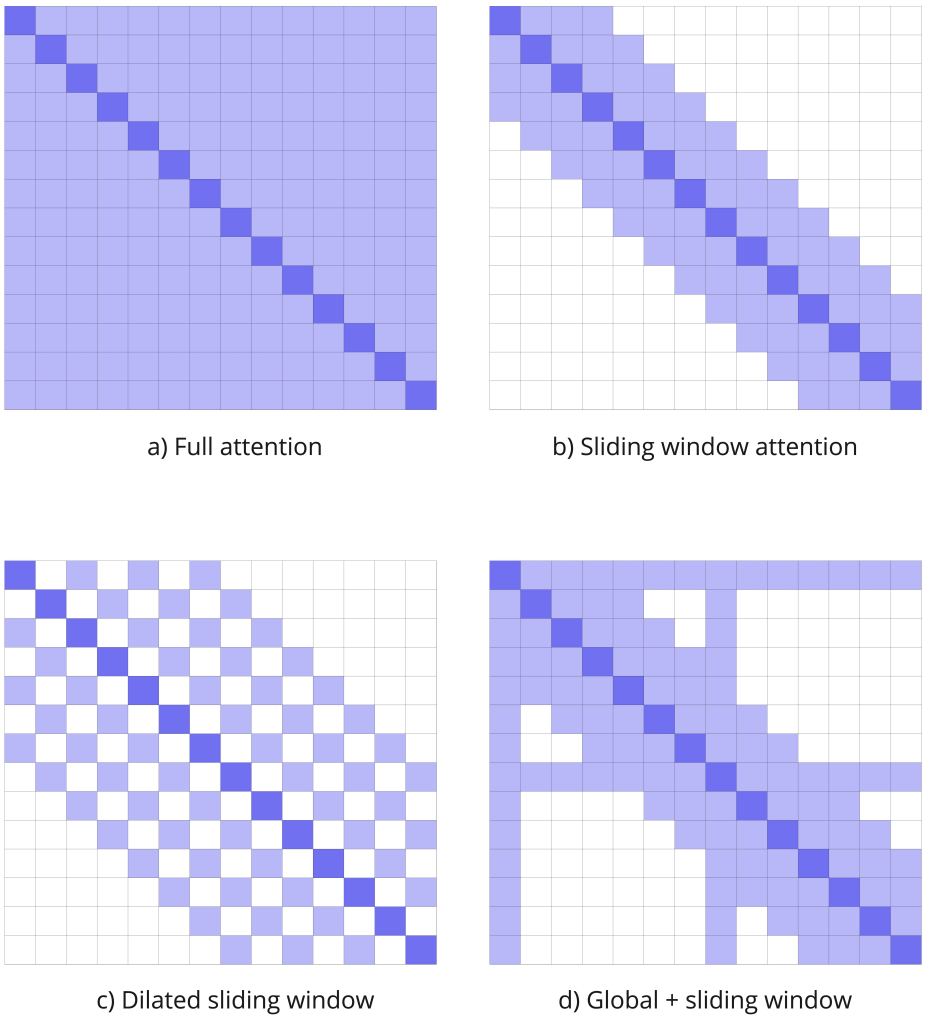


Figure 6: Full self-attention pattern and the attention patterns in Longformer.

token. As a result of multiple stacked layers where this fixed-size attention window is applied to each layer, we obtain a receptive field. The receptive field's top layers have access to all input locations and, thus, can build representations incorporating all available information. Let w be a fixed window size. Each token attends to $\frac{1}{2}w$ tokens on each side. This behavior can be seen in Figure 6 b). The computational complexity of the sliding window attention is $O(n \times w)$. If w is fixed for all layers, then in a transformer with L layers, the receptive field size at the top layer is $L \times w$.

One can increase the receptive field by dilating the sliding window without increasing the computational complexity. The dilated sliding window has gaps of size d. If w and d are fixed for all layers, then in a Transformer with L layers, the receptive field size at the top layer is $L \times w \times d$. The dilated sliding window allows the top layer to reach more tokens, even for small values of d. This behavior can be seen in Figure 6 c). In multi-headed attention, each attention head computes a different attention score. In Longformer, attention heads had different dilation configurations, which was found

to improve performance. Some heads were set to have no dilation, i.e., to focus on local context, whereas others were set with dilation to focus on longer context.

Figure 6 d) shows an example of a sliding window of attention with global attention at a few tokens at custom locations. In Longformer, the windowed and dilated attentions are not flexible enough to learn task-specific representations. Hence, global attention is added to a few pre-selected input locations symmetrically. This means that a token with global attention attends to all tokens across the sequence, and all tokens in the sequence attend to it.

LED has both the encoder and decoder Transformer stacks. The encoder uses the same attention pattern as Longformer, combining a windowed local-context self-attention and an end task-motivated global attention. The window size used in LED is 1024. The decoder uses full self-attention to all the encoded tokens and to previously decoded locations. This model scales linearly with the input. Pre-training LED is expensive; thus, LED parameters are initialized from the BART, with no additional pre-training. The number of layers and hidden sizes follows BART's exact architecture. However, BART can process only 1K tokens, and hence, the position embedding matrix in LED is extended to 16K tokens by copying BART's 1K position embeddings 16 times. LED comes in two sizes, LED_{BASE} and $\text{LED}_{\text{LARGE}}$, having 6 and 12 layers in both encoder and decoder stacks, respectively. LED_{BASE} has $L = 6$ layers for both encoder and decoder, a hidden size of $H = 768$, a feed-forward layer of $F = 3072$ in size, and $A = 8$ self-attention heads.

LED was evaluated on the long-input summarization task using the arXiv summarization dataset (Cohan et al. [54]). The results are seen in Table 6.

3.3.5 GPT-3

GPT-3 is an autoregressive language model with 175 billion parameters. GPT-3 was evaluated on over two dozen NLP datasets and several novel tasks designed to test rapid adaptation to tasks unlikely to be directly contained in the training set. The model was trained on 499 billion tokens of CommonCrawl, WebText, English Wikipedia, and two books corpora. The GPT models are general-purpose language models that can perform various tasks, such as writing code, summarizing text, creating content, and extracting data from documents. (Brown et al. [23])

For each task, GPT-3 was evaluated under three conditions: few-shot learning, whereas many demonstrations as will fit into the model's context window are allowed (typically 10 to 100); one-shot learning, where only one demonstration is allowed, and lastly, zero-shot learning, where no demonstrations are allowed and only instruction in natural language is given to the model.

The first GPT-model was introduced in 2018 in a paper "Improving Language Understanding by Generative Pre-Training" by Radford et al. [42]. GPT-1 is a 12-layer decoder-only Transformer with 12 masked self-attention heads and roughly 110M parameters. BooksCorpus dataset, which contains books from various genres, was used to train the language model. The dataset consists of long passages of contiguous text, which allows the model to learn to condition on long-range information. Gaussian Error Linear Unit (GELU) (Lee et al. [44]) was used as an activation function.

On the other hand, the GPT-2 model was released in 2019 in the paper "Language Models are Unsupervised Multitask Learners" by Radford et al. [43]. The model has a total of 1.5 B parameters. It was pre-trained on the BookCorpus dataset and trained on a dataset of 8 million web pages. With GPT-2, it was demonstrated that language models begin to learn NLP tasks such as question answering, machine translation, reading comprehension, and summarization without any explicit supervision. This is a major difference compared to the supervised learning approaches using large amounts of manually labeled data. [43]

All GPT-models have a generative pre-trained Transformer architecture. The models are autoregressive, specifically unidirectional, meaning they are trained to predict the next word in a sentence. The autoregressive model generates text or predictions one token at a time by conditioning each token's generation on the previously generated tokens. In autoregressive models, the probability distribution of the next token is influenced by the entire sequence of previously generated tokens. This means the model considers the left context (tokens to the left of the current token) during generation. The unidirectional model is a specific type of autoregressive model in which the generation of the next token only depends on tokens to the left of the current position. In other words, it considers a "unidirectional" context, typically the left context. This means that a unidirectional model does not consider any tokens to the right of the current token during generation.

Chunking algorithm

The chunking algorithm is important in text processing pipelines built on the GPT-3 architecture. It is responsible for breaking down large text into manageable chunks of size n that can be processed by GPT-3 and then combining the outputs to create a unified result. The choice of chunking algorithm can range from simply splitting the document into smaller sizes to more complex approaches involving multiple NLP models to determine optimal splitting points. Despite its seemingly straightforward nature, the chunking algorithm holds importance as it acts as a variance control for inputs and significantly impacts the accuracy of text summarization.

The significance of a high-quality chunking algorithm stems from its position at the forefront of the pipeline, where changes in its parameters and capabilities have ripple effects on downstream models. Modifying the output structure or appearance of the chunks often necessitates retraining or adjustment of GPT-3 models, as they were initially refined to work with a specific input structure or data size. Failure to address these changes can lead to a decrease in accuracy. Even seemingly minor adjustments, such as altering the chunk size, require refinement of downstream models due to the level of variance the GPT-3 models are accustomed to. For instance, changes in the number of topics discussed in interview transcript chunks can significantly impact the ability of GPT-3 models to summarize diverse chunks accurately.

This thesis will use a simple chunking algorithm that splits a document into smaller sizes. We aim to find the nearest end of a sentence within a range of $0.5 \times n$ and $1.5 \times n$ tokens.

3.3.6 Results on the summarization tasks

We close this section by presenting previous results on the short- and long-input summarization tasks for the models described earlier. First, we consider the long-input summarization task and fine-tuned versions of the following models: PEGASUS-X_{BASE}, and PEGASUS-X_{LARGE}. These models are fine-tuned with arXiv's scientific articles (Cohan et al. [54]) and evaluated on the test set. LED_{LARGE} is not fine-tuned but evaluated on the arXiv test dataset. The dataset size is 215k, the average document length is 4938 words, and the average summary length is 220 (Cohan et al. [54]).

Results for the long-input summarization task are seen in Table 6. The results are gathered from the paper "Investigating Efficiently Extending Transformers for Long-input Summarization" by Phang et al. [7]. The scores presented in Table 6 are ROUGE1-F1, ROUGE2-F1, and ROUGELs-F1. ROUGELs-F1 refers to the ROUGE-Lsum, a variation of the ROUGE-L metric. Instead of evaluating the summary as a whole, ROUGE- Lsum operates at the sentence level using the ROUGE-L calculation method. It then combines these sentence-level results to generate the final score. PEGASUS-X_{BASE} and PEGASUS-X_{LARGE} achieve almost identical results and slightly outperform LED_{LARGE}. These results provide a benchmark for our long-input summarization task with two distinct sets of articles from arXiv, namely RL and LLM datasets.

Furthermore, we present the results for the short-input summarization task with $T5_{SMALL}$ and $PEGASUS_{LARGE}$ in Table 5. $T5_{SMALL}$ was fine-tuned for a short-input summarization task. A non-anonymized version of the CNN/Daily Mail dataset (See et al. [8]) was used to fine-tune and test the model. On the other hand, arXiv's scientific articles dataset (Cohan et al. [54]) was used to fine-tune and evaluate $PEGASUS_{LARGE}$. For $PEGASUS_{LARGE}$, ROUGEL-F1 was reported instead of ROUGELs-F1 (marked with *).

Hyperparameters for fine-tuning $T5_{SMALL}$, $PEGASUS_{LARGE}$, PEGASUS-X_{BASE}, and PEGASUS-X_{LARGE} are seen in Table 4. LR and BS refer to learning rate and batch size, respectively. The number of steps in fine-tuning is referred to as n_s. Additionally, $l^{\text{max}}_{\text{input}}$ and $l^{\text{max}}_{\text{target}}$ refer to the maximum number of tokens that can occur in the tokenized input and the resulting output.

Model	Dataset	LR	BS	n_s	$l^{\text{max}}_{\text{input}}$	$l^{\text{max}}_{\text{target}}$
$T5_{SMALL}$	CNN/Daily Mail	0.001	128	2^{18}	512	-
$PEGASUS_{LARGE}$	arXiv	5e-4	256	50k	512	256
PEGASUS-X_{BASE}	arXiv	8e-4	64	92.5k	16384	256
PEGASUS-X_{LARGE}	arXiv	8e-4	64	92.5k	16384	256

Table 4: Hyperparameters of the fine-tuning.

Table 5: Comparison of short-input summarization task with different LLMs.

Model	Dataset	ROUGE1-F1	ROUGE2-F1	ROUGELs-F1
$\texttt{T5}_{\texttt{SMALL}}$	CNN/Daily Mail	0.411	0.196	0.384
$\texttt{PEGASUS}_{\texttt{LARGE}}$	arXiv	0.447	0.172	0.257*

Table 6: Comparison of long-input summarization task with different LLMs.

Model	Dataset	ROUGE1-F1	ROUGE2-F1	ROUGELs-F1
$\texttt{PEGASUS-X}_{\texttt{BASE}}$	arXiv	0.494	0.216	0.440
$\texttt{PEGASUS-X}_{\texttt{LARGE}}$	arXiv	**0.500**	**0.218**	**0.446**
$\texttt{LED}_{\texttt{LARGE}}$	arXiv	0.466	0.196	0.418

3.4 Fine-tuning

T5, PEGASUS, PEGASUS-X, and LED were fine-tuned for the summarization task of scientific articles. The RL dataset was used as training, validation, and testing material to fine-tune these models. LLM dataset was utilized only once: a fine-tuned version of PEGASUS-X was trained to summarize LLM-related articles. Fine-tuning scripts for each model were written in Python. The process starts by loading and defining the training, validation, and test datasets. Then, the tokenizer, responsible for preparing the inputs for a model, is loaded. Training and validation data is fed to the tokenizer. The base model is loaded, and hyperparameters for fine-tuning are selected. Tokenized training and validation data is fed to the base model, and the model is fine-tuned.

During the fine-tuning phase, the model is exposed to the labeled training and validation datasets. The training set is used to update the parameters of the pre-trained model, while the validation set helps fine-tune hyperparameters and avoid overfitting. The fine-tuning process involves forward and backward passes through the neural network of the model. For each training example (article-summary pair), the model generates a summary, compares it to the ground truth summary using the cross-entropy loss, and then backpropagates the error through the network to adjust the model's parameters. The model's weights are adjusted using the AdamW, i.e., Adam with decoupled weight decay optimizer (Loshchilov et al. [65]) to minimize the cross-entropy loss. This process is repeated until the cross-entropy loss does not decrease anymore. Then, the fine-tuned model is saved, and the performance is measured against the test dataset. The generated summaries are compared to the original summaries in the test dataset. This comparison uses the evaluation metrics defined in Section Evaluation Metrics 2.5.

Model parameters are seen in Table 7. Size refers to the number of parameters in a base model. For LED, the number of parameters was not reported in the original paper, and thus it is left empty. L refers to the number of layers in a model's encoder and decoder. H, F, and A refer to the hidden size, the size of the feed-forward layer, and the number of self-attention heads, respectively. The models are detailed in Section

Models in our summarization task 3.3. Table 8 shows the hyperparameters chosen for each fine-tuning process. LR, WD, and BS refer to learning rate, weight decay, and batch size, respectively. The number of epochs is referred to as n_e.

In machine learning, weight decay, learning rate, epoch, and batch size are pivotal for effective model training. The learning rate is a critical hyperparameter determining the step size during the fine-tuning phase, influencing the model's convergence. If the learning rate is too high, the model might overshoot the minimum and fail to converge. If it's too low, the model might take a long time to converge or get stuck in a local minimum. The learning rate in each fine-tuning process was chosen based on the recommendations in the papers where the corresponding models were described.

Weight decay is a regularization technique that helps prevent overfitting by penalizing large weights in the model. It involves adding a penalty term to the loss function based on the magnitude of the weights in the model. The regularization term discourages the model from assigning excessively large weights to any particular feature. This helps in creating a simpler model that generalizes better to unseen data. The weight decay was set to 0.01 for each model that was fine-tuned.

Epochs represent the number of times the entire training dataset is processed during model training. Multiple passes over the dataset allow the model to learn from the data, refining its parameters to improve performance. The number of epochs is a hyperparameter that needs to be chosen based on the specific problem and dataset. In our case, the number of epochs was determined by the cross-entropy loss that is calculated for training and validation datasets after each epoch. The fine-tuning was terminated when the cross-entropy loss started to increase or stabilize for the validation set. From Table 8, we note that fine-tuning `LED` required significantly fewer epochs than other models.

Batch size, another crucial hyperparameter, defines the number of training examples processed in one iteration. It affects the trade-off between computational efficiency and parameter update frequency, with small batches introducing more noise but potentially aiding generalization and large batches offering computational efficiency but possibly slower convergence. Tuning these parameters is essential for achieving optimal model performance in diverse machine-learning applications. The batch size was set to one or two in each fine-tuning process as larger numbers led to out-of-memory (OOM).

FE is an abbreviation for "Frozen encoder". If an encoder of a model is frozen during fine-tuning, it is marked with "x" in Table 8. Freezing the encoder parameters before fine-tuning decreases the number of adjustable parameters. This approach may be suboptimal when it comes to the performance of the resulting fine-tuned model, but with a limited amount of memory, this was found to be an effective way to avoid OOM issues.

PEFT is an abbreviation for Parameter-Efficient Fine-Tuning. PEFT is a library for efficiently adapting pre-trained language models to various downstream applications without fine-tuning all the model's parameters. PEFT was first introduced in 2022 by Mangrulkar et al. in [52]. In PEFT, we only fine-tune a small number of model parameters. This significantly decreases computational and storage costs, as fine-tuning large-scale pre-trained language models is costly. Recent PEFT techniques have achieved comparable performance to traditional full fine-tuning. (Mangrulkar et

al. [52]) If PEFT is used before fine-tuning, an "x" is marked in Table 8. The only model where PEFT was used is $\texttt{T5}_{\texttt{SMALL}}$. It would have been beneficial to use PEFT with other models, but it was impossible since the support covers only T5 and BART models. Furthermore, the PEFT model is required to fit into a single GPU. $\texttt{T5}_{\texttt{SMALL}}$ was the only model fulfilling this requirement.

On the other hand, $l^{\text{max}}_{\text{input}}$ and $l^{\text{max}}_{\text{target}}$ refer to the maximum number of tokens that can occur in the tokenized input and the resulting output. In fine-tuning $\texttt{PEGASUS}_{\texttt{LARGE}}$ and $\texttt{PEGASUS-X}_{\texttt{BASE}}$, the target length for the model output was not set. This means we do not explicitly specify the maximum number of tokens allowed in the model output. $l^{\text{max}}_{\text{input}}$ for $\texttt{T5}_{\texttt{SMALL}}$, $\texttt{PEGASUS}_{\texttt{LARGE}}$, and $\texttt{PEGASUS-X}_{\texttt{BASE}}$ was set to the maximum number of tokens that the model can process. Two approaches were tested for LED models. In $\texttt{LED-16k}_{\texttt{BASE}}$, $l^{\text{max}}_{\text{input}}$ was set to 16384, but in $\texttt{LED-8k}_{\texttt{BASE}}$ and $\texttt{LED-8k}_{\texttt{LARGE}}$ it was set to 8192.

As the number of tokens processed by the model increases, the amount of memory also increases. This should be taken into account when deciding the value for $l^{\text{max}}_{\text{input}}$. If our dataset that is used to fine-tune the model contains articles that are, on average, closer to 8192 than 16384, we should probably set $l^{\text{max}}_{\text{input}}$ to 8192. This means we only consider 8192 tokens, and tokens exceeding this limit are cut off. In our case, the cutting is done from left to right, meaning that we take 8192 tokens starting from the beginning of the article.

Model	Size	L	H	F	A
$\texttt{T5}_{\texttt{SMALL}}$	60M	6	512	2048	8
$\texttt{PEGASUS}_{\texttt{LARGE}}$	568M	16	1024	4096	16
$\texttt{PEGASUS-X}_{\texttt{BASE}}$	272M	12	768	3072	16
$\texttt{LED}_{\texttt{BASE}}$		6	768	3072	8
$\texttt{LED}_{\texttt{LARGE}}$		12	1024	4096	16
GPT-3	175B	96	4096	4096	96

Table 7: Model parameters.

Model	LR	WD	BS	n_e	FE	PEFT	$l^{\text{max}}_{\text{input}}$	$l^{\text{max}}_{\text{target}}$
$\texttt{T5}_{\texttt{SMALL}}$	3e-4	0.01	1	48		x	8192	512
$\texttt{PEGASUS}_{\texttt{LARGE}}$	5e-5	0.01	2	116	x		1024	-
$\texttt{PEGASUS-X}_{\texttt{BASE}}$	5e-5	0.01	2	23	x		16384	-
$\texttt{LED-16k}_{\texttt{BASE}}$	1e-4	0.01	1	4	x		16384	1024
$\texttt{LED-8k}_{\texttt{BASE}}$	1e-4	0.01	1	3			8192	1024
$\texttt{LED-8k}_{\texttt{LARGE}}$	1e-4	0.01	1	3	x		8192	1024

Table 8: Hyperparameters in fine-tuning.

Model fine-tuning was run in Amazon Sagemaker. Amazon SageMaker provides machine learning capabilities to prepare, build, train, and deploy ML models efficiently.

Two instances, `ml.g5.24xlarge` and `ml.g5.2xlarge`, were utilized in fine-tuning. The details of the instances are in Table 9. The instance `ml.g5.2xlarge` was used in fine-tuning $\texttt{T5}_{\texttt{SMALL}}$. The other models were fine-tuned using the instance `ml.g5.24xlarge`. The main difference between these instances is the amount of memory available. The number of GPUs increases the costs. Thus, `ml.g5.2xlarge` is significantly cheaper.

Instance	GPU model	vCPU	GPU	GiB	Price per hour
`ml.g5.2xlarge`	NVIDIA A10G	8	1	32	12.73$
`ml.g5.24xlarge`	NVIDIA A10G	96	4	384	1.895$

Table 9: Instance parameters.

3.5 Prompt engineering

A zero-shot learning approach with prompt engineering was applied to `GPT-3`. Model parameters for `GPT-3` can be seen in Table 7. The main difference to fine-tuning is that we don't update the model's weights. This approach doesn't require any training. Instead, we feed the articles to the model one by one with a prompt that contains instructions for the summarization task. We do not give any examples of the articles and the corresponding summaries, thus the zero-shot learning occurs at inference.

The GPT model used was `gpt-3.5-turbo-1106`, and the embedding model was `text-embedding-ada-002`. Starting from the 6th of November 2023, the model `gpt-3.5-turbo-1106` started to support a 16k context window for the prompt and the answer, i.e., $l^{\text{max}}_{\text{input}} + l^{\text{max}}_{\text{target}} = 16384$ tokens. Before that, the maximum context window for the prompt and the answer was 4096 tokens. Thus, the need for a chunking algorithm presented below was crucial before the new updates to the context window were released. We will later refer to `gpt-3.5-turbo-1106` as `GPT-3`.

A custom chunking algorithm was created to process lengthy texts appearing in the RL dataset's test set. The chunking algorithm is responsible for breaking down the article into smaller chunks of size n that are further processed by `GPT-3`. The chunk size used was $n = 1000$, and the temperature was set to 0 in both the creation of the final summary and the creation of the summary for each chunk to minimize the possibility of hallucination. The temperature is a parameter responsible for the degree of randomness in the model's output.

The behavior of the chunking algorithm is explained below.

1. **Chunk the document into n token chunks** The idea is to split a text into smaller chunks of size n, preferably ending at the end of a sentence. We first encode the whole text into tokens. Then, we aim to find the nearest end of a sentence within a range of $0.5 \times n$ and $1.5 \times n$ tokens. To find the end of a sentence, we decode the tokens and check for a full stop or a new line character. If no end of sentence is found, we use n tokens as the chunk size. After creating the chunks, these chunks are decoded back into text.

2. **Process the chunks in parallel** In this part, we apply a prompt to each chunk of text and return a summarized version. The prompt used to guide the model is """Summarize this text from an academic paper. Extract any key points with reasoning. Content:""". Temperature is set to 0 to avoid hallucination.

3. **Summarize the chunks into an overall summary** The chunks are summarized into an overall summary. The prompt used to guide the model with the creation of the final summary is """Summarize this collection of key points extracted from an academic paper in a single paragraph. The summary length must NOT exceed the word limit of $\{summary\_len\}$. Key points: $\{results\}$""", where $summary\_len$ is the length of the original summary and $results$ contains the summarized chunks. Again, the temperature is set to 0 to avoid hallucinations. The word limit for a generated text is thus set to $l_{\text{target}}^{\max} = \{summary\_len\}$.

The pricing of `GPT-3` is based on the tokens. For this model, the price for the inputs is $0.0010 per 1K tokens. The price for the outputs is $0.0020 per 1K tokens.

4 Results

This section presents the results for the long-input summarization task with the RL dataset and the LLM dataset. First, we consider the fine-tuning approach. The training data of the RL dataset is used to fine-tune six models. Furthermore, one model is fine-tuned with the training data of the LLM dataset. The fine-tuning allows the model to adapt for summarising articles from the domains of reinforcement learning (RL) or large language models (LLM). We present the results for the fine-tuned models and their corresponding base versions without fine-tuning. Then, the results for prompting approach with `GPT-3` are presented. The prompting approach is tested using the test data from the RL dataset. Lastly, we answer the research questions posed in Section 1.1. Section Discussion and conclusions 5 analyzes the results in more detail.

The performance of the fine-tuned models and those without fine-tuning was assessed using numerical evaluation metrics. The models that were fine-tuned with the training data of the RL dataset were evaluated using the test data of the RL dataset. Similarly, the model that was fine-tuned with the training data of the LLM dataset was evaluated using the test data of the LLM dataset. The evaluation metrics were ROUGE, BLEU, BERTScore, and NIST. Here, we only report the following ROUGE metrics: ROUGE1-F1, ROUGE2-F1, ROUGEL-F1. ROUGE was chosen as the main evaluation metric due to its popularity. Using ROUGE, we can also compare our models' performance to other published models that were fine-tuned for summarization tasks, as they usually report only ROUGE metrics. The ROUGE scores for both base and fine-tuned models are seen in Table 10. In Table 10, the column name "FT" is an abbreviation for fine-tuning. If the model is fine-tuned, it is marked with an "x" in the corresponding row in the column "FT". The column "Dataset" refers to the dataset that is used to evaluate and/or fine-tune the model's performance. The ROUGE scores are calculated for the test data of the dataset that is marked in the column "Dataset".

$\texttt{T5}_{\texttt{SMALL}}$ was fine-tuned for summarizing articles from the domain of reinforcement learning. The context window was set to 8192 tokens for the tokenized articles. The cross-entropy loss was used to determine when the fine-tuning was completed. The fine-tuning was terminated when the cross-entropy loss started to increase or stabilize for the validation set of the RL dataset. Both fine-tuned and base models were evaluated using the test dataset of the RL dataset. Without fine-tuning $\texttt{T5}_{\texttt{SMALL}}$, the following results are obtained for ROUGE1-F1, ROUGE2-F1, and ROUGEL-F1: 0.193, 0.037, 0.132. For the fine-tuned $\texttt{T5}_{\texttt{SMALL}}$, the ROUGE1-F1, ROUGE2-F1, and ROUGEL-F1 scores are 0.374, 0.116, and 0.215, respectively. The ROUGE metrics are better for the fine-tuned version of $\texttt{T5}_{\texttt{SMALL}}$. However, the non-fine-tuned version of $\texttt{T5}_{\texttt{SMALL}}$ outperforms other base models that were evaluated using the RL test dataset.

$\texttt{PEGASUS}_{\texttt{LARGE}}$ was fine-tuned for summarizing articles from the domain of reinforcement learning. The training data of the RL dataset was used to fine-tune the model. The encoder parameters of the model were frozen to save memory. In $\texttt{PEGASUS}_{\texttt{LARGE}}$, the maximum number of tokens that can be taken into account from each input is only 1024 tokens. In the fine-tuning process, the number of epochs was 116, considerably larger than those in other models' fine-tuning. This is probably due to the small context window of 1024 tokens and the relatively small learning rate of

5e-5. Both fine-tuned and base versions of $\texttt{PEGASUS}_{\texttt{LARGE}}$ were evaluated using the test dataset of the RL dataset. Without fine-tuning $\texttt{PEGASUS}_{\texttt{LARGE}}$ achieves ROUGE1-F1, ROUGE2-F1, and ROUGEL-F1 scores of 0.257, 0.076, and 0.145, respectively. With the fine-tuned version, we obtain the following ROUGE1-F1, ROUGE2-F1, and ROUGEL-F1 scores: 0.412, 0.130, and 0.219. The fine-tuned version clearly outperforms the non-fine-tuned version of $\texttt{PEGASUS}_{\texttt{LARGE}}$.

Two versions of $\texttt{PEGASUS-X}_{\texttt{BASE}}$ were fine-tuned for the summarization task. The first version was fine-tuned with the training data of the LLM dataset and the other with the training data of the RL dataset. In both fine-tuning processes, the encoder parameters were frozen to save memory, and the number of epochs was 23. In $\texttt{PEGASUS-X}_{\texttt{BASE}}$, the context window is of 16k tokens. Thus, we can fit the full-length articles as input for the model without cutting them. Let us first consider $\texttt{PEGASUS-X}_{\texttt{BASE}}$ that was fine-tuned with the training data of the RL dataset. The following ROUGE1-F1, ROUGE2-F1, and ROUGEL-F1 scores are achieved for the fine-tuned model with the test data of the RL dataset: 0.389, 0.126, 0.211. Without fine-tuning, $\texttt{PEGASUS-X}_{\texttt{BASE}}$ achieves the ROUGE1-F1, ROUGE2-F1, and ROUGEL-F1 scores of 0.017, 0.003, and 0.015, respectively, for the test data of the RL dataset. Similar results are obtained when we consider the LLM dataset. When the base model $\texttt{PEGASUS-X}_{\texttt{BASE}}$ is evaluated using the test data of the LLM dataset, ROUGE1-F1, ROUGE2-F1, and ROUGEL-F1 scores are 0.017, 0.003, and 0.014. After fine-tuning $\texttt{PEGASUS-X}_{\texttt{BASE}}$ with the training data of the LLM dataset, we obtain the following ROUGE1-F1, ROUGE2-F1, and ROUGEL-F1 scores for the test data of the LLM dataset: 0.382, 0.129, and 0.214.

Then, three instances of the LED model were fine-tuned for summarizing articles from the reinforcement learning domain. LED can be loaded in different sizes. We used both the base and large versions to see if the performance increases when the model with a larger number of parameters is used. Furthermore, we tested the base version $\texttt{LED}_{\texttt{BASE}}$ with context sizes of 8k and 16k. These three LED models are referred as $\texttt{LED-16k}_{\texttt{BASE}}$, $\texttt{LED-8k}_{\texttt{BASE}}$, and $\texttt{LED-8k}_{\texttt{LARGE}}$, where the 16k and the 8k refer to the used context size. These models were fine-tuned using the training data of the RL dataset, and the models' performance was evaluated on the test data of the RL dataset. During the fine-tuning of $\texttt{LED-16k}_{\texttt{BASE}}$, and $\texttt{LED-8k}_{\texttt{LARGE}}$, the encoder parameters were frozen. The performance of these fine-tuned models and their corresponding non-fine-tuned versions are seen in Table 10. The fine-tuned version of $\texttt{LED-8k}_{\texttt{BASE}}$ outperforms the fine-tuned $\texttt{LED-16k}_{\texttt{BASE}}$. The fine-tuned $\texttt{LED-8k}_{\texttt{BASE}}$ achieves ROUGE1-F1, ROUGE2-F1, and ROUGEL-F1 scores of 0.465, 0.182, and 0.267, respectively, for the test data of the RL dataset. On the other hand, ROUGE1-F1, ROUGE2-F1, and ROUGEL-F1 scores for the fine-tuned $\texttt{LED-16k}_{\texttt{BASE}}$ are 0.430, 0.148, and 0.231. The difference in ROUGE scores can be due to the average length of the articles in the RL dataset (Table 2). The average word count for the RL dataset is 5776, which is approximately 7702 tokens according to OpenAI Token Calculator in [53]. Thus, on average, the context window of 8k tokens should be enough for most articles. When the context window is set to 16k tokens, the number of padded tokens is approximately 8k, which may cause unnecessary weight updates. Surprisingly, the fine-tuned $\texttt{LED-8k}_{\texttt{LARGE}}$ did not perform better than the fine-tuned $\texttt{LED-8k}_{\texttt{BASE}}$, even

though the number of parameters in LED-8k$_{\text{LARGE}}$ is larger. This probably derives from the freezing of the encoder parameters of LED-8k$_{\text{LARGE}}$ during fine-tuning as the number of weights that can be updated decreases.

Lastly, GPT-3 was used to test the prompting approach with the test data of the RL dataset. In prompting, we only give the context, i.e., the article, for the model and ask the model to summarize the article. With the chunk size of $n = 1000$ and temperature of 0, we obtained the following ROUGE1-F1, ROUGE2-F1, and ROUGEL-F1 scores for the test data of the RL dataset: 0.434, 0.135, and 0.220, respectively.

The ROUGE1-F1, ROUGE2-F1, and ROUGEL-F1 scores calculated for the test data of the RL dataset are plotted in Figures 7, 8, and 9, respectively. The "ft" in Figures 7, 8, and 9 refers to a fine-tuned model. The scores for each model are plotted in increasing order. Based on the ROUGE scores, the three best-performing models are the fine-tuned versions of LED-16k$_{\text{BASE}}$, LED-8k$_{\text{BASE}}$, and LED-8k$_{\text{LARGE}}$. These models were fine-tuned with the training data of the RL dataset, and evaluated with the test data of the RL dataset. The prompting approach with GPT-3 performed relatively well, being the 4th best-performing model regarding the ROUGE scores. The fine-tuned LED-8k$_{\text{BASE}}$ outperforms all other models.

Table 10: ROUGE metrics for each model using the RL or LLM dataset test set.

Model	FT	Dataset	ROUGE1-F1	ROUGE2-F1	ROUGEL-F1
T5$_{\text{SMALL}}$		RL	0.193	0.037	0.132
T5$_{\text{SMALL}}$	x	RL	0.374	0.116	0.215
PEGASUS$_{\text{LARGE}}$		RL	0.257	0.076	0.145
PEGASUS$_{\text{LARGE}}$	x	RL	0.412	0.130	0.219
PEGASUS-X$_{\text{BASE}}$		RL	0.017	0.003	0.015
PEGASUS-X$_{\text{BASE}}$	x	RL	0.389	0.126	0.211
PEGASUS-X$_{\text{BASE}}$		LLM	0.017	0.003	0.014
PEGASUS-X$_{\text{BASE}}$	x	LLM	0.382	0.129	0.214
LED-16k$_{\text{BASE}}$		RL	0.088	0.023	0.069
LED-16k$_{\text{BASE}}$	x	RL	0.430	0.148	0.231
LED-8k$_{\text{BASE}}$		RL	0.086	0.023	0.067
LED-8k$_{\text{BASE}}$	x	RL	**0.465**	**0.182**	**0.267**
LED-8k$_{\text{LARGE}}$		RL	0.076	0.016	0.063
LED-8k$_{\text{LARGE}}$	x	RL	0.460	0.171	0.246
GPT-3		RL	0.434	0.135	0.220

4.1 Research questions

Let us next discuss the research questions this thesis aims to answer.

1. **How does a fine-tuned model perform in a long-input summarization task compared to the corresponding base model or prompt engineering?**

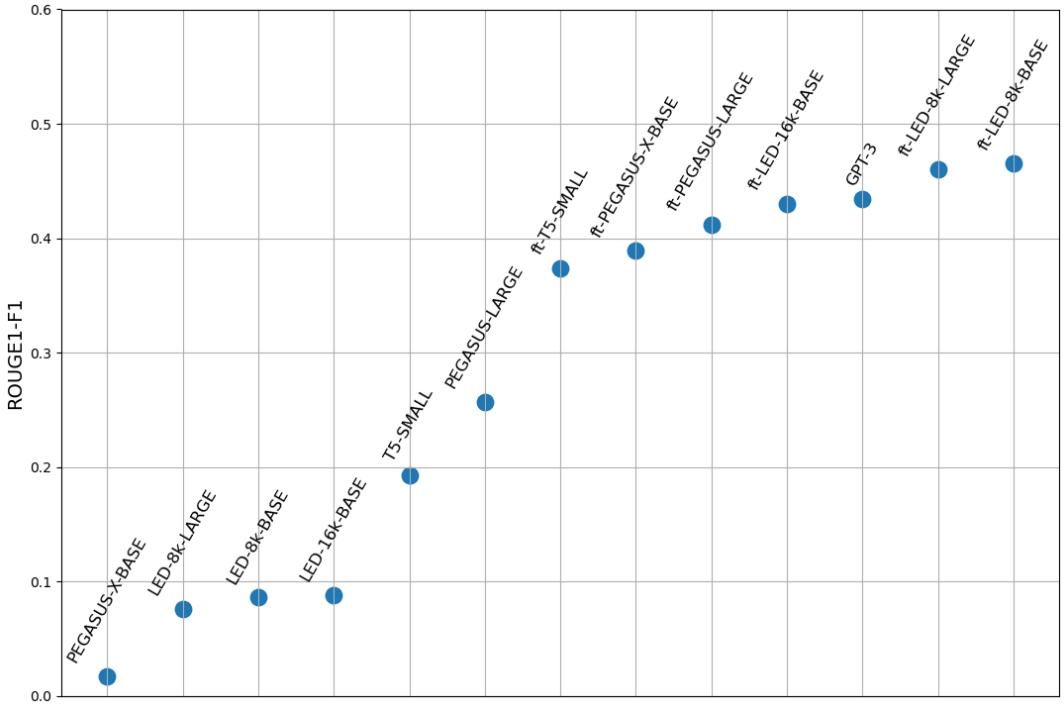


Figure 7: ROUGE1-F1 scores in increasing order. Fine-tuned models are referred to as "ft".

The performance of the fine-tuned models is significantly better than using the corresponding models without fine-tuning. ROUGE metrics for every non-fine-tuned model and their corresponding fine-tuned versions are seen in Table 10. The reported results are run for the test data of the RL or LLM dataset. From the non-fine-tuned models, $\texttt{PEGASUS}_{\texttt{LARGE}}$ achieved the highest scores of 0.257, 0.076, and 0.145 for ROUGE1-F1, ROUGE2-F1, and ROUGEL-F1, respectively. Yet these scores are not even close to the best-performing fine-tuned model, $\texttt{LED-8k}_{\texttt{BASE}}$. This model achieved scores of 0.465, 0.182, and 0.267 for ROUGE1-F1, ROUGE2-F1, and ROUGEL-F1.

The comparison of ROUGE metrics reveals distinctions between the performance of the prompt engineering approach with GPT-3 and the leading fine-tuned model, $\texttt{LED-8k}_{\texttt{BASE}}$. However, the difference is not as remarkable as when comparing the fine-tuned models and their corresponding base models. ROUGE metrics differ between the prompt engineering with GPT-3 and the leading fine-tuned model $\texttt{LED-8k}_{\texttt{BASE}}$. The difference between ROUGE2-F1 and ROUGEL-F1 values of GPT-3 and fine-tuned $\texttt{LED-8k}_{\texttt{BASE}}$ is significant. The fine-tuned $\texttt{LED-8k}_{\texttt{BASE}}$ achieves 0.182 for ROUGE2-F1, whereas the score is only 0.135 for GPT-3. Similarly, ROUGEL-F1 is 0.267 for the fine-tuned $\texttt{LED-8k}_{\texttt{BASE}}$ and 0.220 for GPT-3. ROUGE1-F1 is 0.465 for $\texttt{LED-8k}_{\texttt{BASE}}$, and 0.434 for GPT-3.

Thus, we can conclude that fine-tuning clearly increases the performance of an LLM in the long-input summarization task regarding the ROUGE scores.

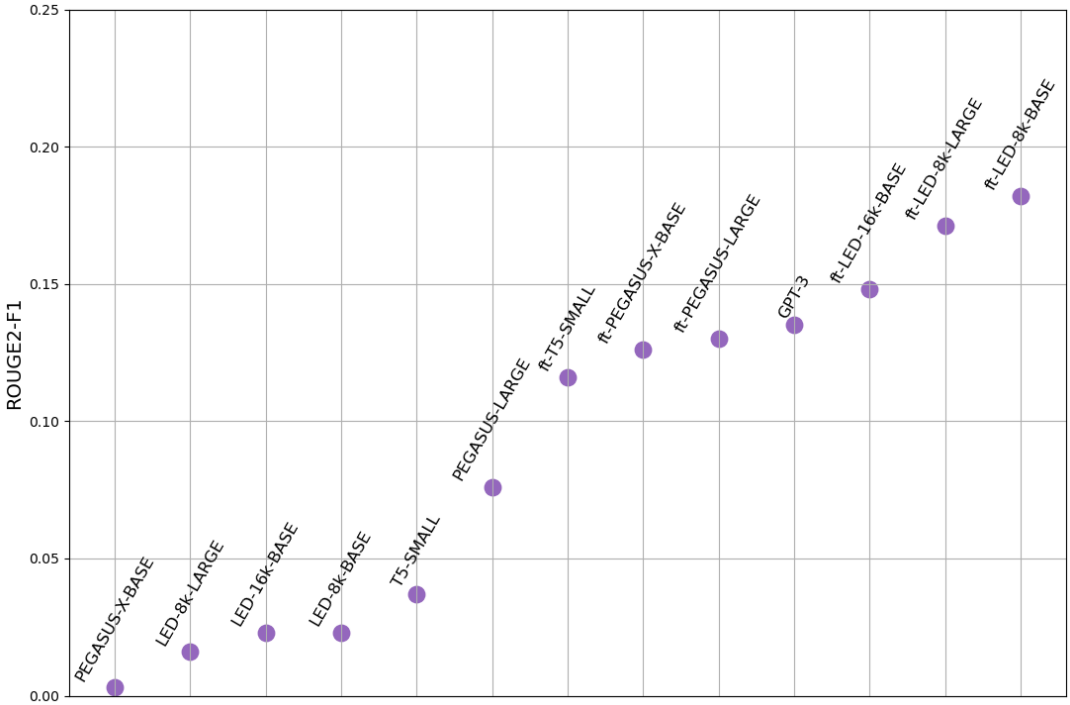


Figure 8: ROUGE2-F1 scores in increasing order. Fine-tuned models are referred to as "ft".

Three top performing fine-tuned models are $\texttt{LED-8k}_{\texttt{BASE}}$, $\texttt{LED-8k}_{\texttt{LARGE}}$, and $\texttt{LED-16k}_{\texttt{BASE}}$. Prompt engineering with `GPT-3` produces great results but this approach cannot outperform the leading fine-tuned models $\texttt{LED-8k}_{\texttt{BASE}}$, and $\texttt{LED-8k}_{\texttt{LARGE}}$.

2. **How do varying hyperparameters and data preprocessing options impact the performance of the models and generated outputs?**

   We answer this research question in two parts. First, we consider the effect of different text preprocessing techniques in fine-tuning. Then, we discuss how varying the hyperparameters affect the prompt engineering with `GPT-3`. Additionally, we test multiple prompts in the prompting approach with `GPT-3`.

   Due to time and space complexities, we could not test multiple hyperparameter sets in fine-tuning, even though it would have been interesting. The hyperparameters, seen in Table 8, were chosen based on the original papers where the models were released (`T5` by Raffel et al. [5], `PEGASUS` by Zhang et al. [4], `PEGASUS-X` by Phang et al. [7], `LED` by Beltagy et al. [3], and `GPT-3` by Radford et al. [42]).

   Text preprocessing in fine-tuning

   Before fine-tuning the models, a custom preprocessing logic was applied to the RL and the LLM datasets. To test how preprocessing of the input text affects fine-tuning, we fine-tune two alternative versions of $\texttt{LED-8k}_{\texttt{BASE}}$

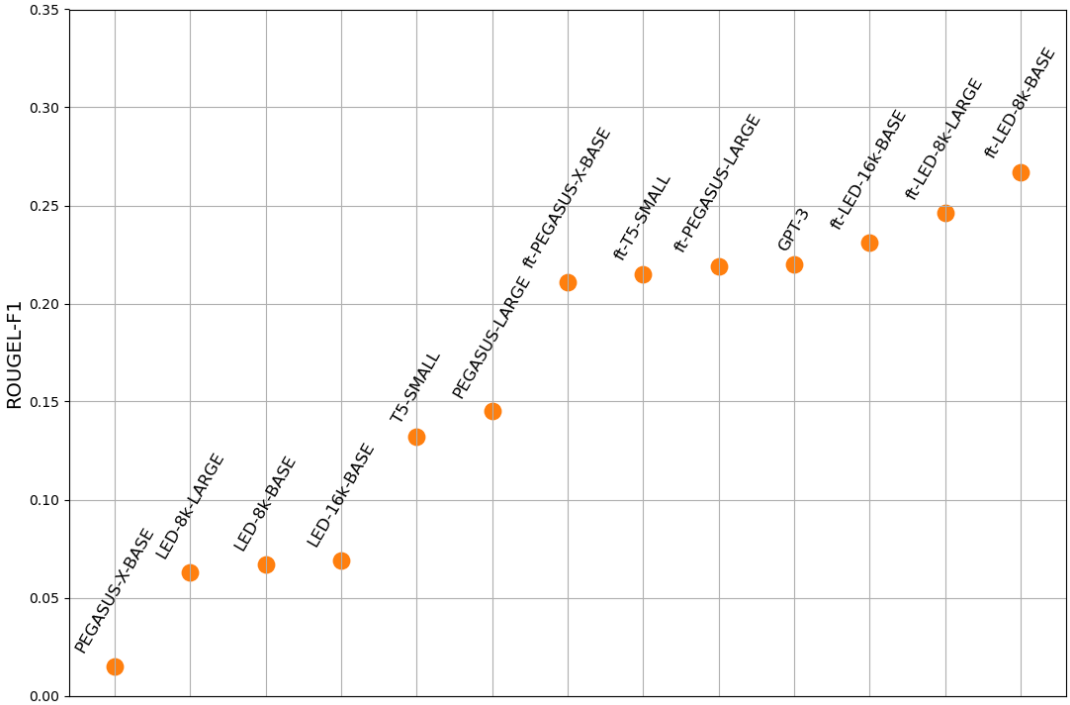


Figure 9: ROUGEL-F1 scores in increasing order. Fine-tuned models are referred to as "ft".

without preprocessing training data and with alternative preprocessing logic. LED-8k$_{\text{BASE}}$ is chosen as its fine-tuned version outperforms other models in terms of ROUGE metrics. The average, minimum, maximum, and median word counts before tokenization and after preprocessing for the inputs (articles) and labels (summaries) for the RL datasets with different preprocessing logics are seen in Table 11. The RL dataset is preprocessed as described in Section Preprocessing 3.2. The RL-2 dataset is processed similarly, but steps 1 and 8 from the original preprocessing logic are omitted. For the RL-3 dataset, abstracts are removed from each article, but other preprocessing steps are discarded. The model that is fine-tuned with the training data of the RL-2 dataset is referred to as LED-2-8k$_{\text{BASE}}$, whereas LED-3-8k$_{\text{BASE}}$ refers to the fine-tuned model with the training data of the RL-3 dataset.

The training data of the RL dataset is used to fine-tune LED-8k$_{\text{BASE}}$, and the model is evaluated using the test data of the RL dataset. Similarly, we fine-tune LED-2-8k$_{\text{BASE}}$ with the training data of the RL-2 dataset and evaluate the model with the test data of the RL-2 dataset. LED-3-8k$_{\text{BASE}}$ refers to the fine-tuned model with the training data of the RL-3 dataset. LED-3-8k$_{\text{BASE}}$ is evaluated with the test data of the RL-3 dataset. The resulting ROUGE scores for the fine-tuned versions of LED-8k$_{\text{BASE}}$ are seen in Table 12. Here, we also present ROUGELs-F1 scores as it alleviates the process of comparing these results to the results seen in Table 6. Out of these three fine-tuned models, LED-2-8k$_{\text{BASE}}$ performs best, achieving the ROUGE1-F1, ROUGE2-F1, ROUGEL-F1, and

ROUGELs-F1 scores of 0.488, 0.196, 0.277, and 0.473, respectively. The fine-tuned LED-3-8k$_{\text{BASE}}$ achieves the ROUGE1-F1, ROUGE2-F1, ROUGEL-F1, and ROUGELs-F1 scores of 0.487, 0.194, 0.275, and 0.471, respectively. The difference in ROUGE scores between LED-2-8k$_{\text{BASE}}$ and LED-3-8k$_{\text{BASE}}$ is diminishing. Surprisingly, the original preprocessing logic results in the lowest ROUGE scores. This result suggests that the amount of preprocessing needed before fine-tuning a model is very case-dependent. In our case, we can conclude that a simpler preprocessing logic results in better performance compared to our original, more complex preprocessing. Based on the results seen in Table 12, we suggest fine-tuning a model for the long-input summarization task in the scientific domain without preprocessing the data. If the results are inadequate, preprocessing logic can be applied step by step to the data.

Hyperparameters and text preprocessing in prompt engineering

In prompt engineering with GPT-3, we can modify the temperature that measures the degree of randomness in the model's output, the chunk size n, and the prompt that contains the instructions for the summarization task. We aim to minimize the model's hallucination. Thus, the temperature in prompt engineering with GPT-3 is set to zero. We do not modify this parameter but instead alter the chunk size n and the prompt. The idea is to study how the generated summaries change when the prompt and the chunk size are altered.

Guiding the model via prompts is a critical factor in achieving high-quality abstracts. We test three different prompts for a small set of articles from the test data of the RL dataset. The size of this small set is 60 articles. Thus, we cannot directly compare the ROUGE scores for this subset of articles to those run for the whole test data of the RL dataset. Also, different values for the chunk size n are tested. Initially, the chunk size is set to $n = 1000$, and the prompt that is given to GPT-3 to summarize each chunk from the article is "Summarize this text from an academic paper. Extract any key points with reasoning. Content:". The prompt for summarizing the chunks into an overall summary is "Summarize this collection of key points extracted from an academic paper in a single paragraph. The summary length must NOT exceed the word limit of $\{summary\_len\}$. Key points: $\{results\}$", where $summary\_len$ is the length of the original summary and $results$ contains the summarized chunks. Let us denote the prompt for a single chunk as prompt$_{\text{c}}$ and the prompt for summarizing all the chunks as prompt$_{\text{all}}$.

We test two alternatives for the prompt prompt$_{\text{all}}$. The first alternative is prompt$_{\text{all-2}}$ = "Create a summary of this list of paragraphs extracted from an academic paper. Paragraphs:$\{results\}$". The other alternative is prompt$_{\text{all-3}}$ = "Write an abstract of this list of paragraphs extracted from a scientific article. Paragraphs:$\{results\}$". These prompts are listed in Table 13. The resulting ROUGE scores are seen in Table 14. The ROUGE scores are calculated for the subset of 60 articles from the test data of the RL dataset.

The highest ROUGE scores are achieved with the prompt prompt$_{\text{all-3}}$. With

the chunk size of 1000 tokens and the prompt $\texttt{prompt}_{\texttt{all-3}}$, ROUGE1-F1, ROUGE2-F1, and ROUGEL-F1 are 0.455, 0.147, and 0.225, respectively. The key difference in the prompt $\texttt{prompt}_{\texttt{all-3}}$ to the two other alternatives is that we use the term "abstract" instead of "summary". Additionally, the length of the prompts $\texttt{prompt}_{\texttt{all-2}}$ and $\texttt{prompt}_{\texttt{all-3}}$ is shorter compared to the original prompt $\texttt{prompt}_{\texttt{all}}$.

Next, we test three alternatives for the original chunk size n of 1000 tokens. We use the prompt $\texttt{prompt}_{\texttt{all-3}}$ as it outperforms the other alternatives as seen in Table 14. The alternative chunk sizes are 1200, 1500, and 2000. The results are in Table 15. The highest ROUGE1-F1 score, 0.455, is obtained when the chunk size is 1000 tokens. However, with a chunk size of 1000 tokens, the ROUGE1-F1 score is only 0.002 higher than the ROUGE1-F1 score with a chunk size of 1500 tokens. The highest ROUGE2-F1 score of 0.152 is obtained when the chunk size is set to 1500. ROUGEL-F1 is 0.225 with the chunk sizes 1000, 1500, and 2000.

We can conclude that the style and wording of the prompt have an effect on the outputs of GPT-3. We were able to increase the ROUGE scores by simplifying and changing the wording of the prompt that is used to guide the model to summarize the articles. The term "abstract" led to better results than using the word "summary" in the prompt. The chunk size also affects the generated summaries by GPT-3, yet the impact is smaller than the correct choice for the prompt.

3. **How to comprehensively evaluate the quality of generated texts?**

ROUGE metrics provide only a limited view of the quality of the generated text. As discussed in Section Limitations 2.5.7, human evaluation is a vital yet time- and resource-intensive part of assessing the quality of generated texts due to language's inherent complexity and subjectivity. While automated metrics, such as ROUGE, offer quantitative measures of certain linguistic aspects, they often fall short of capturing human language's nuanced and contextual nature.

Language is inherently subjective, and assessing qualities like fluency, coherence, creativity, and adherence to style requires a level of understanding that machines may struggle to achieve. Human evaluators bring a depth of comprehension that extends beyond the quantitative metrics. Their ability to grasp the subtleties of language, understand context, and make subjective judgments is invaluable in determining the overall quality of generated content.

Human judgment becomes paramount in tasks like natural language generation, where the goal is to mimic patterns and convey meaning, relevance, and coherence. Automated metrics might excel at measuring surface-level features, such as n-gram overlap. Still, they cannot often discern whether the generated text truly fulfills the intended purpose or meets the expectations of a human audience.

Moreover, language is not a one-size-fits-all phenomenon. Style, tone and even

creative expression can vary widely based on individual preferences and the specific requirements of a given task. Human evaluators can provide insights into these subjective elements, helping to tailor the evaluation process to the application's specific needs. Higher-level cognitive functions such as logical reasoning, common sense understanding, and context awareness significantly evaluate text quality. These elements are often beyond the reach of automated metrics, which makes human evaluators indispensable in ensuring that the generated content aligns with the intended goals and effectively communicates with the target audience.

In conclusion, while automated metrics offer quantitative benchmarks, the depth and breadth of language evaluation demand human involvement. Integrating automated metrics and human judgment provides a more holistic view of text quality. It ensures that language's intricacies, including its subjective and contextual dimensions, are accurately captured and assessed.

Table 11: The average, minimum, maximum, and median word counts before tokenization and after preprocessing for the inputs (articles) and labels (summaries) in RL datasets.

Dataset	Column	Average	Median	Minimum	Maximum
RL	Input	5776	5239	44	58832
RL	Label	155	153	14	297
RL-2	Input	6087	5425	49	71176
RL-2	Label	156	153	14	297
RL-3	Input	8436	7135	455	168253
RL-3	Label	156	153	14	297

Table 12: Comparison of long-input summarization task with different preprocessing logics. Metrics from left to right are ROUGE1-F1, ROUGE2-F1, ROUGEL-F1, and ROUGELs-F1. ROUGE metrics for the test sets are presented.

Model	Dataset	R-1	R-2	R-L	R-Ls
LED-8k$_{\text{BASE}}$	RL	0.465	0.182	0.267	0.450
LED-2-8k$_{\text{BASE}}$	RL-2	**0.488**	**0.196**	**0.277**	**0.473**
LED-3-8k$_{\text{BASE}}$	RL-3	0.487	0.194	0.275	0.471

Table 13: The prompts used in prompt engineering with GPT-3.

$\texttt{prompt}_{\texttt{c}}$	Summarize this text from an academic paper. Extract any key points with reasoning. Content:
$\texttt{prompt}_{\texttt{all}}$	Summarize this collection of key points extracted from an academic paper in a single paragraph. The summary length must NOT exceed the word limit of $\{summary\_len\}$. Key points: $\{results\}$
$\texttt{prompt}_{\texttt{all-2}}$	Create a summary of this list of paragraphs extracted from an academic paper. Paragraphs:$\{results\}$
$\texttt{prompt}_{\texttt{all-3}}$	Write an abstract of this list of paragraphs extracted from a scientific article. Paragraphs:$\{results\}$

Table 14: Comparison of long-input summarization task with GPT-3 and varying prompt.

Model	Prompt	Chunk size	ROUGE1-F1	ROUGE2-F1	ROUGEL-F1
GPT-3	$\texttt{prompt}_{\texttt{all}}$	1000	0.418	0.123	0.209
GPT-3	$\texttt{prompt}_{\texttt{all-2}}$	1000	0.429	0.137	0.214
GPT-3	$\texttt{prompt}_{\texttt{all-3}}$	1000	**0.455**	**0.147**	**0.225**

Table 15: Comparison of long-input summarization task with GPT-3 and varying chunk size n.

Model	Prompt	Chunk size	ROUGE1-F1	ROUGE2-F1	ROUGEL-F1
GPT-3	$\texttt{prompt}_{\texttt{all-3}}$	1000	**0.455**	0.147	**0.225**
GPT-3	$\texttt{prompt}_{\texttt{all-3}}$	1200	0.448	0.145	0.221
GPT-3	$\texttt{prompt}_{\texttt{all-3}}$	1500	0.453	**0.152**	**0.225**
GPT-3	$\texttt{prompt}_{\texttt{all-3}}$	2000	0.448	0.143	**0.225**

5 Discussion and conclusions

We start this section by summarizing the work and analyzing the results we obtained in Section Summary and analysis 5.1. Then, we draw conclusions regarding the research problem in Section Conclusions 5.2. Additionally, we discuss the limitations of using only numerical evaluation metrics in assessing the performance of an LLM in Section Limitations 5.3. Lastly, in Section Recommendations and future work 5.4, recommendations are given, and future work is discussed.

5.1 Summary and analysis

In this thesis, we studied how large language models can be adapted to summarize scientific articles. Two sets of scientific articles from arXiv were collected. The first dataset contained approximately 9000 articles about reinforcement learning. The other dataset was smaller, containing approximately 1000 articles about large language models. These datasets are called the RL and the LLM datasets, respectively. The data was preprocessed before feeding the data to the models.

We utilized pre-trained large language models as a starting point, namely `T5`, `PEGASUS`, `PEGASUS-X`, and `LED`. These pre-trained models were trained on large datasets for various tasks. `T5` is pre-trained with unlabeled data with the objective of teaching the model generalizable knowledge. `PEGASUS` and `PEGASUS-X` are intended to be used for abstractive summarization. `PEGASUS-X` was specifically pre-trained for long-input summarization. `LED` is intended to use in long document generative sequence-to-sequence tasks. Two approaches, fine-tuning and prompt engineering, were applied to the base models. Fine-tuning adapted these models to the task of summarizing scientific articles from the field of reinforcement learning or large language models. In many real-world scenarios, obtaining a large labeled dataset for training a model from scratch may be impractical or expensive. Thus, fine-tuning also allows leveraging pre-existing models trained on large datasets and adapting them to our task with a smaller, task-specific dataset. Prompt engineering was studied as an alternative for fine-tuning. In prompt engineering, only a prompt and an article without examples were provided to the model at inference. Despite having no prior knowledge of the specific content, the model uses its general language understanding and reasoning abilities to generate contextually relevant text.

Six models were fine-tuned using the training data of the RL dataset: $\texttt{T5}_{\texttt{SMALL}}$, $\texttt{PEGASUS}_{\texttt{LARGE}}$, $\texttt{PEGASUS-X}_{\texttt{BASE}}$, $\texttt{LED-16k}_{\texttt{BASE}}$, $\texttt{LED-8k}_{\texttt{BASE}}$, and $\texttt{LED-8k}_{\texttt{LARGE}}$. Another version of $\texttt{PEGASUS-X}_{\texttt{BASE}}$ was fine-tuned using the training data of the LLM dataset. These models were evaluated using the corresponding test data of either the RL or LLM datasets. The prompt engineering approach with the test data of the RL dataset was studied with `GPT-3`.

ROUGE was used as an evaluation metric of the models' performance. Among all the models, including both the fine-tuned and the non-fine-tuned models with the initial preprocessing logic, the fine-tuned $\texttt{LED-8k}_{\texttt{BASE}}$ emerged with the highest ROUGE scores. ROUGE1-, ROUGE2-, ROUGEL-, and ROUGELs-F1 scores for the fine-tuned $\texttt{LED-8k}_{\texttt{BASE}}$ were 0.465, 0.182, 0.267, and 0.450, respectively. The

results for all models are seen in Table 10. Prompt engineering approach with GPT-3 resulted in the following ROUGE1-, ROUGE2-, and ROUGEL-F1 scores for the RL test dataset: 0.434, 0.135, and 0.220, respectively.

To study how data preprocessing affects the fine-tuned model's performance, we ran additional tests for LED-8k$_{\text{BASE}}$ by considering two alternative preprocessing logics for the RL dataset. The resulting datasets with two other preprocessing logics are referred to as the RL-2 and the RL-3 datasets. The RL dataset is preprocessed as described in Section Preprocessing 3.2. The RL-2 dataset was processed similarly, but steps 1 and 8 from the original preprocessing logic were omitted. In the RL-3 dataset, only the abstracts were removed from the articles. The model that is fine-tuned with the training data of the RL-2 dataset is referred to as LED-2-8k$_{\text{BASE}}$, whereas LED-3-8k$_{\text{BASE}}$ refers to the fine-tuned model with the training data of the RL-3 dataset.

It was found that both alternative preprocessing logics improved the ROUGE scores. The scores can be seen in Table 12. With the initial preprocessing logic, the ROUGE1-, ROUGE2-, ROUGEL-, and ROUGELs-F1 scores for the fine-tuned LED-8k$_{\text{BASE}}$ were 0.465, 0.182, 0.267, and 0.450 for the test data of the RL dataset. When lowercasing and removing special characters and numbers from the preprocessing logic were omitted, the ROUGE1-, ROUGE2-, ROUGEL-, and ROUGELs-F1 scores increased moderately to 0.488, 0.196, 0.277, and 0.473 for the fine-tuned LED-2-8k$_{\text{BASE}}$. This model, LED-2-8k$_{\text{BASE}}$, outperforms all the other models adapted to the summarization task of scientific articles from the domains of reinforcement learning or large language models. We note that, at least for the scientific articles, the initial preprocessing logic did not result in the best performance. Unexpectedly, a simpler preprocessing resulted in better performance regarding ROUGE scores.

Furthermore, we studied how the data preprocessing options and different hyperparameters affect the performance of the prompting approach with GPT-3. These approaches were tested for a subset of 60 articles from the test data of the RL dataset. First, we wrote three different prompts that guide the model in the summarization task and kept the chunk size as 1000 tokens. The prompts can be seen in Table 13, and the results are reported in Table 14. The initial prompt prompt$_{\text{all}}$ was "Summarize this collection of key points extracted from an academic paper in a single paragraph. The summary length must NOT exceed the word limit of $\{summary\_len\}$. Key points: $\{results\}$", where $summary\_len$ is the length of the original summary and $results$ contains the summarized chunks. With this prompt and the chunk size of 1000 tokens, the ROUGE1-, ROUGE2-, and ROUGEL-F1 scores for a subset of 60 articles from the test data of the RL dataset were 0.418, 0.123, and 0.209. The following prompt, prompt$_{\text{all-3}}$, produced the best results compared to other alternatives in terms of ROUGE scores: "Write an abstract of this list of paragraphs extracted from a scientific article. Paragraphs:$\{results\}$", where $results$ contains the summarized chunks. With the aforementioned prompt, the ROUGE1-, ROUGE2-, and ROUGEL-F1 scores for the RL test dataset increased to 0.455, 0.147, and 0.229, respectively.

Additionally, we varied the chunk size n used in the chunking algorithm. The prompt prompt$_{\text{all-3}}$ resulted in the highest ROUGE scores with the subset from the test data of the RL dataset. Thus, it was used when varying the chunk size n. Chunk sizes of 1000, 1200, 1500, and 2000 were tested. The results are seen in Table 15. The

highest ROUGE1-F1 score is 0.455, which is obtained with the initial chunk size of 1000. However, the highest ROUGE2-F1 score of 0.152 is obtained when the chunk size is set to 1500. ROUGEL-F1 is 0.225 with the chunk sizes 1000, 1500, and 2000. Thus, it is not straightforward to conclude which value for n is optimal. It can be argued that choosing the chunk size is case-dependent. Furthermore, the chunking algorithm could be improved to find paragraphs from the input text instead of splitting the article into chunks of size n. We may split the text suboptimally with our naive chunking algorithm, presented in Section Prompt Engineering 3.5.

5.2 Conclusions

Based on the ROUGE scores, our fine-tuned models (Table 10) for summarizing scientific articles, namely from RL- or LLM-related topics, can compete with the models fine-tuned for the similar task seen in Table 6. The models, seen in Table 6, are fine-tuned utilizing a large set of scientific articles from arXiv (Cohan et al. [54]), and the task is to summarize scientific articles from various domains. In our case, the task is identical but the datasets used to fine-tune and test the model are smaller in size and more specific, containing articles from either LLM- or RL-related topics. The RL and LLM datasets used in our long-input summarization task contain approximately 9k and 1.5k articles, respectively. In contrast, the arXiv dataset (Cohan et al. [54]), presented in Section Results on the summarization task 3.3.6, contains 215k scientific articles. The models that are fine-tuned with the arXiv dataset (Cohan et al. [54]) may generalize better due to a broader set of topics. However, the summarization task in both cases aims to summarize scientific articles, and thus we can meaningfully compare the resulting ROUGE scores.

Among the models we fine-tuned with the training data of the RL dataset, the best-performing model was the fine-tuned version of $\texttt{LED-8k}_{\texttt{BASE}}$. This fine-tuned model achieves the following ROUGE1-F1, ROUGE2-F1, ROUGEL-F1, and ROUGELs-F1 scores for the test data of the RL dataset: 0.465, 0.182, 0.267, and 0.473 (see Table 12). By simplifying the preprocessing logic, we were able to increase the ROUGE scores for $\texttt{LED-8k}_{\texttt{BASE}}$. The RL-2 dataset refers to a dataset that is preprocessed with simplified logic. Using the training data of the RL-2 dataset, we fine-tuned $\texttt{LED-2-8k}_{\texttt{BASE}}$ that achieves the ROUGE1-F1, ROUGE2-F1, ROUGEL-F1, and ROUGELs-F1 scores of 0.488, 0.196, 0.277, and 0.473, respectively, for the test data of the RL-2 dataset.

On the other hand, among the models that were fine-tuned with the arXiv dataset (Cohan et al. [54]) for the summariztion task, the best-performing model was $\texttt{PEGASUS-X}_{\texttt{LARGE}}$, achieving ROUGE1-F1 of 0.500, ROUGE2-F1 of 0.218, and ROUGELs-F1 of 0.446 for the test data of the arXiv dataset (see Table 6). The fine-tuned $\texttt{PEGASUS-X}_{\texttt{LARGE}}$ has higher ROUGE1-F1 and ROUGE2-F1 scores compared to our best-performing fine-tuned model $\texttt{LED-2-8k}_{\texttt{BASE}}$. However, the fine-tuned $\texttt{LED-2-8k}_{\texttt{BASE}}$ has higher ROUGELs-F1 score than the fine-tuned $\texttt{PEGASUS-X}_{\texttt{LARGE}}$.

Based on ROUGE scores, we can conclude that it is possible to successfully adapt a language model to a summarization task with only a small amount of data. Choosing the correct pre-trained model as a starting point for the fine-tuning is task- and data-dependent. For a long-input summarization task, choosing between

fine-tuning and prompt engineering requires careful consideration of different aspects. In this thesis, fine-tuning led to better results than the prompt engineering approach. However, we studied only the long-input summarization task of scientific articles. In some other domains, prompt engineering may be a better choice. Summarization of scientific articles benefits from fine-tuning because the vocabulary and the structure of a scientific article differ from the unstructured data used in pre-training the LLMs. Furthermore, it was found that a simpler preprocessing logic that omits lowercasing and removing numbers from the data produced better results than a more detailed one.

In fine-tuning, we utilized open-source models that are free to use. The cost of fine-tuning was based on the GPU usage in Amazon Sagemaker. After fine-tuning, we could run the fine-tuned models on inference without cost. The prompt engineering approach with `GPT-3` was, in our case, cheaper than fine-tuning. However, using `GPT-3` is more expensive in the long run. The charges are based on the number of input and output tokens. The cost would quickly increase if we would like to build an application with multiple users.

Yet, we have only analyzed the performance of the fine-tuned and base models on the summarization task relying on the ROUGE scores. Extensive human evaluation would significantly improve the assessment of the generated summaries, but unfortunately, it is out of the scope of this thesis. However, some comments can be given regarding the generated texts. During the evaluation of the models, the generated abstracts from the articles of the test dataset were saved together with their corresponding original abstract and the ROUGE scores. Thus, we can access each article, its abstract, the generated abstracts, and corresponding ROUGE scores by each model shown in Table 10. Skimming through the generated abstracts of the RL-2 test dataset by the leading fine-tuned model $\texttt{LED-2-8k}_{\texttt{BASE}}$ reveals that, in general, this model produces sensible text with correct abbreviations, and the texts consist of coherent sentences with punctuation. Assessing the quality of the generated texts produced by the models that were fine-tuned with the training data of the RL dataset is more challenging. This derives from the original preprocessing logic in which the text is lowercased. Thus, reading the generated text and comparing it to the original abstract becomes tedious. The generated abstracts of the RL test dataset by the worst performing fine-tuned model $\texttt{T5}_{\texttt{SMALL}}$ seem to contain more irrational sentences and repetition compared to the fine-tuned $\texttt{LED-2-8k}_{\texttt{BASE}}$. However, these comments are only based on quick browsing of the generated texts. To dive deeper into the models' differences in producing summaries of scientific articles, a rigorous human evaluation process of the generated texts should be carried out.

5.3 Limitations

There are noteworthy limitations when assessing the performance of LLMs fine-tuned or prompt-engineered for summarizing scientific articles using numerical metrics like ROUGE.

Firstly, ROUGE primarily relies on lexical overlap, emphasizing surface-level similarities. However, this approach may fall short in evaluating the semantic understanding of the model. A high ROUGE score does not necessarily ensure that the

generated summary accurately reflects the intricate scientific concepts presented in the original abstract. Scientific writing often employs domain-specific terminology, and complex language structures, not to mention equations, graphs, and figures. ROUGE cannot fully account for the precision and accuracy of these terms, potentially leading to disparities between the generated summary and the original abstract, particularly in terms of specificity.

Additionally, scientific discourse often involves synonymy and paraphrasing to convey nuanced ideas. Numerical metrics may penalize the model for using different yet valid expressions, possibly underestimating the quality of the generated summaries. The contextual understanding required for scientific articles, including relationships between concepts, may not be fully captured by ROUGE. Scientific domains vary widely, each with unique conventions and writing styles. Generic metrics might not adequately address the specific requirements of different scientific disciplines, resulting in biased evaluations.

Scientific articles frequently include technical details crucial for a comprehensive understanding. Numerical metrics might struggle to evaluate the correctness and accuracy of these details, potentially overlooking critical information in the generated summaries. Furthermore, scientific articles often present novel research contributions, and numerical metrics might not be sensitive to the novelty of the generated content. This limitation could lead to an incomplete evaluation of the impact and significance of the generated summaries. With their domain expertise, human experts are better equipped to evaluate the accuracy and appropriateness of the generated summaries. Relying solely on numerical metrics may not capture the insights of these experts, who can provide valuable qualitative assessments.

Lastly, ROUGE and similar metrics are not designed to assess the cohesiveness and flow of the generated summaries. In scientific writing, the logical progression of ideas is crucial, and deficiencies in this aspect may not be adequately reflected in numerical scores. To address these limitations comprehensively, it is encouraged to supplement numerical metrics with qualitative assessments by domain experts if possible. Human evaluations, particularly from those with expertise in the specific scientific domain, offer valuable insights into the accuracy, coherence, and overall quality of the generated summaries. This combined approach ensures a more thorough evaluation of the LLM's performance in summarizing scientific articles.

5.4 Recommendations and future work

There are a few recommendations that should be taken into account when studying long-input summarization using LLMs. We will next make some recommendations and comment on future work.

The lack of labeled datasets may become an issue if we want to generalize the long-input summarization task to other domains. Scientific articles were chosen as a domain because the abstract is usually a mandatory part of the article. Thus, constructing a labeled dataset is easy. This may not be the case with real-world use cases. For example, a company can easily have thousands of documents from a specific industry without summaries. Fine-tuning an LLM for long-input summarization when only

documents are available, without corresponding summaries, is challenging. We may utilize prompt engineering, but it should be noted that without reference summaries, evaluating the generated texts becomes hard. In this case, human evaluation becomes crucial.

Secondly, scientific articles usually contain images, graphs, tables, and equations, besides the unique vocabulary. Even though the summary usually contains only text, it could be beneficial to leverage the information in tables and equations in fine-tuning or in-context learning. LLMs that integrate information from multiple modalities, such as text and images, already exist. One example is GPT-4 by OpenAI [55]. GPT-4 is a large-scale, multimodal model that can accept image and text inputs and produce text outputs. We cannot directly conclude that this model could process tables or equations, but it is a step forward in solving the problem.

Lastly, our best-performing model, $\texttt{LED-2-8k}_{\texttt{BASE}}$, is fine-tuned only for summarizing the articles from the reinforcement learning domain. We have not tested the model's performance with the articles from different domains. It is likely that the model succeeds in summarizing scientific articles from other domains that are relatively close to reinforcement learning. Yet, this assumption should be confirmed by testing the fine-tuned model with the articles from other domains.

The development of tools and applications utilizing LLMs has been rapid. In the near future, businesses and individuals may have more accessible tools for customizing and fine-tuning pre-trained LLMs for specific tasks, making them more applicable to various industries and use cases. Researchers are likely to continue working on developing more advanced and efficient language models. This includes refining architectures and training techniques and addressing limitations such as biases and ethical concerns. Furthermore, there may be a trend towards creating specialized LLMs for specific domains or industries. This could lead to more accurate and context-aware language understanding in various fields.

References

[1] OpenAI API, "Fine-tuning - Preparing Your Dataset." Docs - Guides - Fine-tuning. https://platform.openai.com/docs/guides/fine-tuning/preparing-your-dataset. (accessed on June 6, 2023)

[2] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, I. Polosukhin, "Attention Is All You Need." *NIPS'17: Proc. of the 31st Int. Conf. on Neural Inf. Process. Syst.*, 2017, vol. 119, pp. 6000–6010.

[3] I. Beltagy, M. E. Peters, A. Cohan, "LongFormer: The Long-Document Transformer." Online article. Available: http://arxiv.org/abs/2004.05150.

[4] J. Zhang, Y. Zhao, M. Saleh, P. J. Liu, "PEGASUS: Pre-training with Extracted Gap-sentences for Abstractive Summarization." *ICML'20: Proc. of the 37th Int. Conf. on Mach. Learn.*, 2020, vol. 119, pp. 11328–11339. DOI: 10.5555/3524938.3525989.

[5] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, P. J. Liu, "Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer." *The J. of Mach. Learn. Res.*, 2020, vol. 21, issue 01, pp. 5485–5551. Also available online: https://jmlr.org/papers/volume21/20-074/20-074.pdf.

[6] M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, L. Zettlemoyer, "BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension." *Proc. of the 58th Annu. Meeting of the Assoc. for Comput. Linguistics*, 2020, pp. 7871–7880. DOI: 10.18653/v1/2020.acl-main.703.

[7] J. Phang, Y. Zhao, P. J. Liu, "Investigating Efficiently Extending Transformers for long-input Summarization." *Proc. of the 2023 Conf. on Empirical Methods in Natural Lang. Process.*, 2023, pp. 3946–3961. DOI: 10.18653/v1/2023.emnlp-main.240.

[8] A. See, P. J. Liu, C. D. Manning, "Get To The Point: Summarization with Pointer-Generator Networks." *J. of Intell. Learn. Syst. and Appl.*, 2018, vol. 10, no. 04, pp. 1–20. DOI: 10.18653/v1/P17-1099.

[9] Y. Tay, M. Dehghani, D. Bahri, D. Metzler, "Efficient Transformers: A Survey." *ACM Comput. Surv.*, 2022, vol. 51, issue 06, pp. 1–28. DOI: 10.1145/3530811.

[10] J. Devlin, M. Chang, K. Lee, K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding." *North American Chapter of the Assoc. for Comput. Linguistics*, 2019, pp. 4171–4186. DOI: 10.18653/v1/N19-1423.

[11] X. Wu, R. Duan, J. Ni, "Unveiling Security, Privacy, and Ethical Concerns of ChatGPT." *J. of Inf. and Intell.*, 2023, pp. 1–14. DOI: 10.1016/j.jiixd.2023.10.007.

[12] M. Zaheer, G. Guruganesh, A. Dubey, J. Ainslie, C. Alberti, S. Ontanon, P. Pham, A. Ravula, Q. Wang, L. Yang, A. Ahmed, "Big Bird: Transformers for Longer Sequences." *NIPS'20: Proc. of the 34th Int. Conf. on Neural Inf. Process. Syst.*, 2020, pp. 17283–17297.

[13] K. Choromanski, V. Likhosherstov, D. Dohan, X. Song, A. Gane, T. Sarlos, P. Hawkins, J. Davis, A. Mohiuddin, L. Kaiser, D. Belanger, L. Colwell, A. Weller, "Rethinking Attention with Performers." *Int. Conf. on Learn. Representations 2021 Oral*, 2021. Available: https://openreview.net/forum?id=Ua6zuk0WRH.

[14] J. Su, Y. Lu, S. Pan, A. Murtadha, B. Wen, Y. Liu, "RoFormer: Enhanced Transformer With Rotary Position Embedding." *Neurocomput.*, 2023, vol. 568, pp- 1–14. DOI: 10.1016/j.neucom.2023.127063.

[15] A. Wang, Y. Pruksachatkun, N. Nangia, A. Singh, J. Michael, F. Hill, O. Levy, and S. Bowman, "Superglue: A stickier benchmark for general-purpose language understanding systems." *NIPS'19: Proc. of the 33rd Int. Conf. on Neural Inform. Process. Syst.*, 2019, vol. 32, pp. 3266–3280. DOI: 10.5555/3454287.3454581.

[16] H. Y. Koh, J. Ju, M. Liu, S. Pan, "An Empirical Survey on Long Document Summarization: Datasets, Models and Metrics." *ACM Comput. Surv.*, 2022, vol. 55, issue 08, pp. 1–35. DOI: 10.1145/3545176.

[17] W. S. El-Kassas, C. R. Salama, A. A. Rafea, H. K. Mohamed, "Automatic text summarization: A comprehensive survey." *Expert Syst. with Appl.* 2021, vol. 165, pp. 1–46. Online publication. Also available in print. DOI: 10.1016/j.eswa.2020.113679.

[18] P. Liu, W. Yuan, J. Fu, Z. Jiang, H. Hayashi, G. Neubig, "Pre-train, Prompt, and Predict: A Systematic Survey of Prompting Methods in Natural Language Processing." *ACM Comput. Surv.*, 2023, vol. 55, issue 09, pp. 1–35. DOI: 10.1145/3560815.

[19] E. Brill, R. J. Mooney, "An Overview of Empirical Natural Language Processing." *The AI Mag.*. 1997, vol. 18, no. 4, pp. 13–24. Also available in print. DOI: 10.1609/aimag.v18i4.1318.

[20] Wikipedia contributors, "Large language model." *Wikipedia, The Free Encyclopedia*, 2023. Available: https://en.wikipedia.org/wiki/Large_language_model. (accessed Oct. 17, 2023)

[21] Wikipedia contributors, "Recurrent neural network." *Wikipedia, The Free Encyclopedia*, 2023. Available: https://en.wikipedia.org/wiki/Recurrent_neural_network. (accessed Oct. 17, 2023)

[22] Wikipedia contributors, "Convolutional neural network." *Wikipedia, The Free Encyclopedia*, 2023. Available: https://en.wikipedia.org/wiki/Convolutional_neural_network. (accessed Oct. 17, 2023)

[23] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, D. Amodei, "Language Models are Few-Shot Learners." *NIPS'20: Proc. of the 34th Int. Conf. on Neural Inf. Process. Syst.*, 2020, pp. 1877–1901.

[24] E. Hovy, "Text Summarization." *The Oxford Handbook of Comput. Linguistics,* 2005, pp. 583–598. DOI: 10.1093/oxfordhb/9780199276349.013.0032.

[25] N. Reimers, I. Gurevych, "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks." *Conf. on Empirical Methods in Natural Lang. Process.*, 2019, pp. 3982–3992. DOI: 10.18653/v1/D19-1410.

[26] M. Mosbach, T. Pimentel, S. Ravfogel, D. Klakow, Y. Elazar, "Few-shot Fine-tuning vs. In-context Learning: A Fair Comparison and Evaluation." Online article. Available: http://arxiv.org/abs/2305.16938.

[27] A. Awadalla, M. Wortsman, G. Ilharco, S. Min, I. Magnusson, H. Hajishirzi, L. Schmidt, "Exploring The Landscape of Distributional Robustness for Question Answering Models." *Findings of the Assoc. for Comput. Linguistics: EMNLP 2022*, pp. 5971–5987. Also available online: https://aclanthology.org/2022.findings-emnlp.441.

[28] Wikipedia contributors, "Self-supervised learning." *Wikipedia, The Free Encyclopedia*, 2023. Available: https://en.wikipedia.org/wiki/Self-supervised_learning. (accessed Oct. 19, 2023)

[29] Wikipedia contributors, "Weak learning." *Wikipedia, The Free Encyclopedia*, 2023. Available: https://en.wikipedia.org/wiki/Weak_supervision. (accessed Oct. 19, 2023)

[30] Wikipedia contributors, "Directed acyclic graph." *Wikipedia, The Free Encyclopedia*, 2023. Available: https://en.wikipedia.org/wiki/Directed_acyclic_graph. (accessed Oct. 19, 2023)

[31] Wikipedia contributors, "Long short-term memory." *Wikipedia, The Free Encyclopedia*, 2023. Available: https://en.wikipedia.org/wiki/Long_short-term_memory. (accessed Oct. 19, 2023)

[32] F. A. Gers, J. Schmidhuber, F. Cummins, "Learning to Forget: Continual Prediction with LSTM." *Neural Comput.,* 2000, pp. 2451–2471. DOI: 10.1162/089976600300015015.

[33] Wikipedia contributors, "Feedforward neural network." *Wikipedia, The Free Encyclopedia*, 2023. Available: https://en.wikipedia.org/wiki/Feedforward_neural_network. (accessed 19.10.2023)

[34] Wikipedia contributors, "Multilayer perceptron." *Wikipedia, The Free Encyclopedia*, 2023. Available: https://en.wikipedia.org/wiki/Multilayer_perceptron. (accessed 19.10.2023)

[35] arXiv contributors, *arXiv*. https://arxiv.org/. (accessed 20.10.2023)

[36] Y. Yu, S. Zuo, H. Jiang, W. Ren, T. Zhao, C. Zhang, "Fine-Tuning Pre-trained Language Model with Weak Supervision: A Contrastive-Regularized Self-Training Approach." *Proc. of the 2021 Conf. of the North American Chapter of the Assoc. for Comput. Linguistics: Human Lang. Technol.*, 2021, pp. 1063–1077. DOI: 10.18653/v1/2021.naacl-main.84.

[37] M. Sokolova, N. Japkowicz, S. Szpakowicz, "Beyond Accuracy, F-Score and ROC: A Family of Discriminant Measures for Performance Evaluation." *AI 2006: Advances in Artificial Intell., Lecture Notes in Comput. Sci.,* 2006, vol. 4304, pp. 1015–1021. DOI: 10.1007/11941439_114.

[38] C. Lin, "ROUGE: A Package for Automatic Evaluation of Summaries." *Text Summarization Branches Out*, 2004, pp. 74–81. Assoc. for Comput. Linguistics. Also available online: https://aclanthology.org/W04-1013.pdf.

[39] K. Papineni, S. Roukos, T. Ward, W. Zhu, "Bleu: a Method for Automatic Evaluation of Machine Translation." *Proc. of the 40th Annu. Meeting of the Assoc. for Comput. Linguistics*, 2002, pp. 311–318. Also available online: https://aclanthology.org/P02-1040.

[40] T. Zhang, V. Kishore, F. Wu, K. Q. Weinberger, Y. Artzi, "BERTScore: Evaluating Text Generation with BERT." *ICLR 2020*, pp. 1–43.

[41] G. Doddington, "Automatic Evaluation of Machine Translation Quality Using N-gram Co-Occurrence Statistics." *HLT '02: Proc. of the second int. conf. on Human Lang. Technol. Res.*, 2002, pp. 138–145. DOI: 10.5555/1289189.1289273.

[42] A. Radford., K. Narasimhan, T. Salimans, I. Sutskever, "Improving Language Understanding by Generative Pre-Training." Online article. Available: https://api.semanticscholar.org/CorpusID:49313245.

[43] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, "Language Models are Unsupervised Multitask Learners." Online article. Available: https://www.bibsonomy.org/bibtex/2c692ad1906553fce788d166721041c70/msteininger.

[44] M. Lee, "Mathematical Analysis and Performance Evaluation of the GELU Activation Function in Deep Learning." *J. of Math.*, 2023, vol. 2023, pp. 1–13. DOI: 10.1155/2023/4229924.

[45] Wikipedia contributors, "Artificial Neural Network." *Wikipedia, The Free Encyclopedia*, 2023. Available: https://en.wikipedia.org/wiki/Artificial_neural_network. (accessed Nov. 27, 2023)

[46] E. M. Bender, T. Gebru, A. McMillan-Major, S. Shmitchell, "On the Dangers of Stochastic Parrots: Can Language Models Be Too Big?." *Proc. of the 2021 ACM Conf. on Fairness, Accountability, and Transparency*, 2021, pp. 610–623. DOI: 10.1145/3442188.3445922.

[47] V. Nair, G. E. Hinton, "Rectified linear units improve restricted boltzmann machines." *ICML'10: Proc. of the 27th Int. Conf. on Int. Conf. on Mach. Learn.*, 2010, pp. 807–814. Also available online: https://api.semanticscholar.org/CorpusID:15539264.

[48] Z. Ji, N. Lee, R. Frieske, T. Yu, D. Su, Y. Xu, E. Ishii, Y. Bang, A. Madotto, P. Fung, "Survey of hallucination in natural language generation." *ACM Comput. Surv.*, vol. 55, issue 12, pp. 1–38. DOI: 10.1145/3571730.

[49] S. Gehrmann, E. Clark, T. Sellam, "Repairing the Cracked Foundation: A Survey of Obstacles in Evaluation Practices for Generated Text." *J. of Artif. Intell. Res.*, 2023, vol. 77, pp. 103–166. DOI: 10.1613/jair.1.13715.

[50] J. Risch, T. Möller, J. Gutsch, M. Pietsch, "Semantic Answer Similarity for Evaluating Question Answering Models." *Proc. of the 3rd Workshop on Mach. Reading for Question Answering*, 2021, pp. 149–157. Also available online: https://aclanthology.org/2021.mrqa-1.15.

[51] A. Tabassum, R. R. Patil, "A Survey on Text Pre-Processing & Feature Extraction Techniques in Natural Language Processing." *Int. Res. J. of Eng. and Technol. (IRJET)*, 2020, vol. 07, issue 06, pp. 4864–4867.

[52] S. Mangrulkar, S. Gugger, L. Debut, Y. Belkada, S. Paul, B. Bossan, "PEFT: State-of-the-art Parameter-Efficient Fine-Tuning methods." 2022. Available: https://github.com/huggingface/peft. (accessed Dec. 1, 2023)

[53] OpenAI Token Calculator, "OpenAI Token Calculator." Available: https://www.gptcostcalculator.com/open-ai-token-calculator. (accessed Dec. 5, 2023)

[54] A. Cohan, F. Dernoncourt, D. S. Kim, T. Bui, S. Kim, W. Chang, N. Goharian, "A discourse-aware attention model for abstractive summarization of long documents." *In Proc. of the 2018 Conf. of the North American Chapter of the Assoc. for Comput. Linguistics: Human Lang. Technol.*, 2018, vol. 2 (Short Papers), pp. 615–621. Also available online: https://www.aclweb.org/anthology/N18-2097.

[55] OpenAI (2023), "GPT-4 Technical Report." Online article. Available: http://arxiv.org/abs/2303.08774.

[56] K. M. Hermann, T. Kocisky, E. Grefenstette, L. Espeholt, W. Kay, M. Suleyman, P. Blunsom, "Teaching machines to read and comprehend." *NIPS'15: Proc. of the 28th Int. Conf. on Neural Inf. Process. Syst.*, 2015, vol. 01, pp. 1693–1701.

[57] H. Zhou, X. Wan, L. Proleev, D. Mincu, J. Chen, K. Heller, R. Subhrajit, "Batch Calibration: Rethinking Calibration for In-context Learning and Prompt Engineering." *Submitted to The Twelfth Int. Conf. on Learn. Representations*, under review, 2023. Also available online: https://openreview.net/forum?id=L3FHMoKZcS.

[58] P. Kouris, G. Alexandridis, A. Stafylopatis, "Abstractive Text Summarization: Enhancing Sequence-to-Sequence Models Using Word Sense Disambiguation and Semantic Content Generalization." *Comput. Linguistics*, 2021, vol. 47, issue 04, pp. 813–859. Also available online: https://aclanthology.org/2021.cl-4.27.

[59] M. Mohri, A. Afshin Rostamizadeh, A. Talwalkar, "Foundations of Machine Learning." *Adaptive Comput. and Mach. Learn.*, MIT Press, 2nd edition, 2018.

[60] X. Liu, F. Zhang, Z. Hou, L. Mian, Z. Wang, J. Zhang, J. Tang, "Self-Supervised Learning: Generative or Contrastive." *IEEE Trans. on Knowledge and Data Eng.*, 2023, vol. 35, no. 1, pp. 857–876. DOI: 10.1109/TKDE.2021.3090866.

[61] X. Qiu, T. Sun, Y. Xu, Y. Shao, N. Dai, X. Huang, "Pre-trained Models for Natural Language Processing: A Survey." *Sci. China Technological Sciences*, 2020, vol. 63, pp. 1872–1897. Online article. DOI: 10.1007/s11431-020-1647-3.

[62] P. Daugherty, H. J. Wilson, K. Narain, "Generative AI Will Enhance - Not Erase - Customer Service Jobs." hbr.org. https://hbr.org/2023/03/generative-ai-will-enhance-not-erase-customer-service-jobs. (accessed Dec. 27, 2023)

[63] J. Chung, C. Gulcehre, K. Cho, Y. Bengio, "Gated Feedback Recurrent Neural Networks." *Proc. of the 32nd Int. Conf. on Mach. Learn.*, 2015, vol. 37, pp. 2067–2075.

[64] A. Zhang, Z. C. Lipton, M. Li, A. J. Smola, "Dive into Deep Learning." Cambridge University Press. https://D2L.ai. (accessed Dec. 27, 2023)

[65] I. Loshchilov, F. Hutter, "Decoupled Weight Decay Regularization." *ICLR 2019 Conf.*, pp. 1–8.

A Example Model Outputs

This appendix contains example model outputs from the fine-tuned models. Analyzing the generated abstracts and comparing the generated and reference texts is based on the writer's abilities. Thus, comprehensive fact-checking is omitted from the analysis.

A.1 LED-2-8k-BASE

In Table A1, there is an arbitrarily selected example of a generated abstract from the test data of the RL dataset by the fine-tuned LED-2-8k$_{\text{BASE}}$. LED-2-8k$_{\text{BASE}}$ is fine-tuned with the training data of the RL dataset. The abbreviations in the generated text are correct. The generated summary introduces the proposed technique, Variational Hierarchical Reinforcement Learning (VHRL), without delving deeply into the specific challenges open-domain dialog systems face. Furthermore, the generated summary repeats the statement about VHRL improving human judgments of conversational quality, appearing twice in the summary.

Table A1: Generated abstract from the test data of the RL dataset by the fine-tuned LED-2-8k$_{\text{BASE}}$.

ROUGE1-F1	0.432
ROUGE2-F1	0.170
ROUGEL-F1	0.304
ROUGELs-F1	0.412
Original	Open-domain dialog generation is a challenging problem; maximum likelihood training can lead to repetitive outputs, models have difficulty tracking long-term conversational goals, and training on standard movie or online datasets may lead to the generation of inappropriate, biased, or offensive text. Reinforcement Learning (RL) is a powerful framework that could potentially address these issues, for example by allowing a dialog model to optimize for reducing toxicity and repetitiveness. However, previous approaches which apply RL to open-domain dialog generation do so at the word level, making it difficult for the model to learn proper credit assignment for long-term conversational rewards. In this paper, we propose a novel approach to hierarchical reinforcement learning, VHRL, which uses policy gradients to tune the utterance-level embedding of a variational sequence model. This hierarchical approach provides greater flexibility for learning long-term, conversational rewards. We use self-play and RL to optimize for a set of human-centered conversation metrics, and show that our approach provides significant improvements – in terms of both human evaluation and automatic metrics – over state-of-the-art dialog models, including Transformers.
Generated	Open-domain dialog systems have been a fundamental challenge in artificial intelligence (AI). Current generative models for dialog suffer from several shortcomings that limit their usefulness in the real world. In this paper, we propose a novel technique, Variational Hierarchical Reinforcement Learning (VHRL), which leverages policy gradients to adjust the prior probability distribution of the latent variational model learned at the utterance level of a hierarchical variety of dialog models. We show that VHRL improves human judgments of conversational quality above state-of-the-art dialog architectures, including Transformer-based models. Our evaluation shows that VHRL improves human judgments of conversational quality above state-of-the-art dialog architectures, including Transformer.

In Table A2, this thesis is summarized by the fine-tuned LED-2-8k$_{\text{BASE}}$. Even though the thesis deals with long-input summarization using large language models, and LED-2-8k$_{\text{BASE}}$ is fine-tuned with the training data of the RL dataset that contains articles from the reinforcement learning domain, the ROUGE scores are relatively high. This thesis is also considerably longer than the articles in the RL dataset. Thus, the context window of 8k tokens is probably too narrow. In the generated text, there is no mention of using large language models to solve the task. The generated

text mentions models but does not specify the term. The following sentence in the generated is ambiguous: "The short-input summarization task is formulated as a long-input summarization task that requires the model to focus on critical elements, summarize articles, and selecting suitable values for the model's outputs.".

Table A2: Generated abstract from this thesis by the fine-tuned $\texttt{LED-2-8k}_{\texttt{BASE}}$.

ROUGE1-F1	0.409
ROUGE2-F1	0.123
ROUGEL-F1	0.192
ROUGELs-F1	0.380
Original	Large language models (LLMs) have shown remarkable capabilities in various natural language processing tasks. However, their output may not always meet the specific requirements or domain knowledge needed. The generated text may lack coherence or factuality, especially in summarization tasks with longer inputs. The increasing demand for automated summarization, coupled with the complexity of summarizing scientific content, presents a unique challenge. This thesis will focus on the long-input summarization task of scientific articles using LLMs. The thesis employs abstractive summarization techniques and explores two prominent strategies: fine-tuning LLMs and prompting. Fine-tuning involves adapting pre-trained models to the summarization task, leveraging their vast pre-trained knowledge, while prompting relies on structured instructions to guide LLMs in generating summaries without altering their weights. The research comprehensively analyzes these approaches, evaluating their strengths and weaknesses regarding summary quality, computational efficiency, and adaptability to the scientific domain. Utilizing relatively small datasets from arXiv, the thesis showcases successful fine-tuning even with a limited amount of data, examines the impact of text preprocessing on fine-tuning, and optimizes prompt engineering through multiple prompts and a custom chunking algorithm. The limitations of numerical evaluation metrics in assessing text quality are critically analyzed. The research aims to provide valuable insights into long-input summarization, offering guidance on the effectiveness of fine-tuning and prompting strategies to enhance LLMs' capabilities in processing extensive and intricate textual documents. The results of this study show that the fine-tuning strategy outperforms the prompting approach in the long-input summarization task of scientific articles. Furthermore, comparing non-fine-tuned and fine-tuned LLMs reveals that fine-tuning is a crucial step in using an LLM to summarize scientific articles, even with models pre-trained for summarization. We emphasize the limitations of using only numerical evaluation metrics in assessing the quality of generated texts and conclude that human evaluation is a vital part of ensuring the factuality and coherence of the generated summaries.
Generated	Long-input summarization deals with the challenging task of summarizing extended scientific articles from a given text. This task is challenging for models as they must navigate issues of fluency, redundancy, and semantic coherence when current state-of-the-art models for a normal document cannot be implemented effectively due to hardware and model limitations. In this paper, we tackle the long-input summarization task by formulating the task as a short-input summary generation task. The short-input summarization task is formulated as a long-input summarization task that requires the model to focus on critical elements, summarize articles, and selecting suitable values for the model's outputs. We propose two approaches for tackling the long-input summarization task: a prompt containing instructions for summarization and a prompt to summarize scientific documents. The prompt is used to guide the model's behavior in generating a summary from a given text. The evaluation will consider various aspects, including computational efficiency, computational efficiency, models' adaptability to a scientific domain, and the need for labeled data.

A.2 GPT-3

In Table A3, there is an arbitrarily selected example of a generated abstract from the test data of the RL dataset by `GPT-3` with $\texttt{prompt}_{\texttt{all-3}}$ and chunk size of 1500. The original summary is more concise and focuses on key points, providing a high-level overview of the main contributions and findings of the paper. The generated summary is more detailed and delves into specific aspects of the paper, such as the taxonomy of heterogeneous systems, the evaluation of performance and resilience, and the support from various organizations. Furthermore, the original summary focuses on the technical aspects of the paper, such as the introduction of "hetgppo" and its use of a

graph neural network for inter-agent communication. It also mentions the taxonomical overview presented in the paper. The generated summary includes more technical details and discusses the mechanisms suggested by the authors to measure and tune the degree of policy heterogeneity.

Table A3: Generated abstract from the test data of the RL dataset by `GPT-3`.

ROUGE1-F1	0.413
ROUGE2-F1	0.117
ROUGEL-F1	0.215
Original	cooperative multi-robot tasks can benefit from heterogeneity in the robots' physical and behavioral traits. in spite of this, traditional multi-agent reinforcement learning (marl) frameworks lack the ability to explicitly accommodate policy heterogeneity, and typically constrain agents to share neural network parameters. this enforced homogeneity limits application in cases where the tasks benefit from heterogeneous behaviors. in this paper, we crystallize the role of heterogeneity in marl policies. towards this end, we introduce heterogeneous graph neural network proximal policy optimization (hetgppo), a paradigm for training heterogeneous marl policies that leverages a graph neural network for differentiable inter-agent communication. hetgppo allows communicating agents to learn heterogeneous behaviors while enabling fully decentralized training in partially observable environments. we complement this with a taxonomical overview that exposes more heterogeneity classes than previously identified. to motivate the need for our model, we present a characterization of techniques that homogeneous models can leverage to emulate heterogeneous behavior, and show how this ""apparent heterogeneity"" is brittle in real-world conditions. through simulations and real-world experiments, we show that: (i) when homogeneous methods fail due to strong heterogeneous requirements, hetgppo succeeds, and, (ii) when homogeneous methods are able to learn apparently heterogeneous behaviors, hetgppo achieves higher resilience to both training and deployment noise.
Generated	This academic paper discusses the use of heterogeneous multi-robot systems in various applications and the challenges in synthesizing optimal decentralized policies for these systems. It introduces a taxonomy of heterogeneous systems and presents a framework called hetgppo for training heterogeneous multi-agent reinforcement learning policies. The paper also evaluates the performance and resilience of heterogeneous policies compared to homogeneous ones in cooperative multi-robot tasks. The taxonomy classifies heterogeneity into physical and behavioral, with subclasses for each. The paper also reviews the current state of the art in the area of heterogeneous multi-robot/agent systems and classifies related works according to the taxonomy. The paper concludes by highlighting the need for a shared and structured taxonomy of heterogeneous systems and the potential for further research in this area. The paper also discusses the benefits of policy heterogeneity for multi-robot tasks in simulation and the real world, demonstrating that heterogeneous models are able to solve tasks more efficiently and with greater resilience compared to homogeneous models. The paper also highlights the limitations of homogeneous models in emulating heterogeneous behavior and the brittleness of inferred indexing. The authors suggest the development of mechanisms to measure and actively tune the degree of policy heterogeneity in multi-robot teams for better control of trade-offs between sample efficiency and resilience. The research was supported by various organizations including the European Research Council and the US Army Research Laboratory.

A.3 PEGASUS-X-BASE

In Table A4, there is an arbitrarily selected example of a generated abstract from the test data of the RL dataset by the fine-tuned `PEGASUS-X`$_{\texttt{BASE}}$. The model is fine-tuned with the training data of the RL dataset. In the generated abstract, it is stated that the framework studied is "prediction-then-optimize" (colored in blue). Still, the actual term used in the original text is "predict-then-optimize". Furthermore, in the generated text, there is a sentence that refers to a figure (marked in violet) but it is unclear what the figure is and how it relates to the sentence. Lastly, colored in red, the following characters exist twice [„„] that shouldn't be part of the text. Besides these minor details, the generated text seems coherent.

Table A4: Generated abstract from the test data of the RL dataset by the fine-tuned PEGASUS-X_{BASE}.

ROUGE1-F1	0.527
ROUGE2-F1	0.201
ROUGEL-F1	0.222
Original	in the predict-then-optimize framework, the objective is to train a predictive model, mapping from environment features to parameters of an optimization problem, which maximizes decision quality when the optimization is subsequently solved. recent work on decision-focused learning shows that embedding the optimization problem in the training pipeline can improve decision quality and help generalize better to unseen tasks compared to relying on an intermediate loss function for evaluating prediction quality. we study the predict-then-optimize framework in the context of sequential decision problems (formulated as mdps) that are solved via reinforcement learning. in particular, we are given environment features and a set of trajectories from training mdps, which we use to train a predictive model that generalizes to unseen test mdps without trajectories. two significant computational challenges arise in applying decision-focused learning to mdps: (i) large state and action spaces make it infeasible for existing techniques to differentiate through mdp problems, and (ii) the high-dimensional policy space, as parameterized by a neural network, makes differentiating through a policy expensive. we resolve the first challenge by sampling provably unbiased derivatives to approximate and differentiate through optimality conditions, and the second challenge by using a low-rank approximation to the high-dimensional sample-based derivatives. we implement both bellman–based and policy gradient–based decision-focused learning on three different mdp problems with missing parameters, and show that decision-focused learning performs better in generalization to unseen tasks.
Generated	we study the prediction-then-optimize framework in sequential decision problems, formulated as markov decision processes (mdps), with unknown parameters. in particular, at training time, we are given trajectories and environment features from "training mdps." our goal is to learn a predictive model which maps from environment features to missing parameters based on these trajectories that generalizes to unseen test mdps that have features, but not trajectories. the resulting "predicted" training and test mdps are solved using deep reinforcement learning (rl) algorithms, yielding policies that are then evaluated by offline off-policy evaluation (ope) as shown in figure. this fully offline setting is motivated by real-world applications such as wildlife conservation and tuberculosis treatment where no simulator is available. however, such domains offer past ranger patrol trajectories and environmental features of individual locations from conservation parks for generalization to other unpatrolled areas. these settings differ from those considered in transfer-rl [„] and meta-rl [„] because we generalize across different mdps by explicitly predicting the mapping function from features to missing mdps parameters, while transfer/meta rl achieve generalization by learning hidden representation of different mdps implicitly with trajectories. the main contribution of this paper is to extend the decision-focused learning approach to mdps with unknown parameters, embedding the mdp problems in the predictive model training pipeline. to perform this embedding, we study two common types of optimality conditions in a bellman-based approach where mean-squared bellman error is minimized, and a policy gradient-based approach where the expected cumulative reward is maximized. we convert these optimality conditions into their corresponding karush–kuhn–tucker (kkt) conditions, where we can backpropagate through the embedding by differentiating through the kkt conditions. we empirically test our decision-focused algorithms on three a grid world with unknown rewards, and snare-finding and tuberculosis treatment problems where transition probabilities are unknown.

A.4 PEGASUS-LARGE

In Table A5, there is an example of a generated abstract from the test data of the RL dataset by the fine-tuned $PEGASUS_{LARGE}$. This model is fine-tuned on the training data of the RL dataset. ROUGE scores and the original abstract are also shown. Unnecessary repetitions in the generated abstract are highlighted in red and blue. The generated text contains two abbreviations colored in violet, "vvse" and "vce", which both appear to be incorrect. The correct abbreviation is "vcse" which stands for value-conditional state entropy, and can be seen in the original abstract. This generated abstract acts as an example of a text that has relatively good ROUGE scores but lacks coherence and has incorrect abbreviations and repetition.

Table A6 shows another example of a generated abstract from the RL-test dataset

Table A5: Generated abstract from the test data of the RL dataset by the fine-tuned $\text{PEGASUS}_{\text{LARGE}}$.

ROUGE1-F1	0.524
ROUGE2-F1	0.194
ROUGEL-F1	0.303
Original	a promising technique for exploration is to maximize the entropy of visited state distribution, i.e., state entropy, by encouraging uniform coverage of visited state space. while it has been effective for an unsupervised setup, it tends to struggle in a supervised setup with a task reward, where an agent prefers to visit high-value states to exploit the task reward. such a preference can cause an imbalance between the distributions of high-value states and low-value states, which biases exploration towards low-value state regions as a result of the state entropy increasing when the distribution becomes more uniform. this issue is exacerbated when high-value states are narrowly distributed within the state space, making it difficult for the agent to complete the tasks. in this paper, we present a novel exploration technique that maximizes the value-conditional state entropy, which separately estimates the state entropies that are conditioned on the value estimates of each state, then maximizes their average. by only considering the visited states with similar value estimates for computing the intrinsic bonus, our method prevents the distribution of low-value states from affecting exploration around high-value states, and vice versa. we demonstrate that the proposed alternative to the state entropy baseline significantly accelerates various reinforcement learning algorithms across a variety of tasks within minigrid, deepmind control suite, and meta-world benchmarks. source code is available at https://sites.google.com/view/rl-vcse.
Generated	the aim of value-conditional state entropy exploration is to train agents that maximize the sum of task reward and/or intrinsic reward based on a state entropy estimate. we show that this approach can accelerate reinforcement learning when the distribution of high-value states and/or low-value states is uniform. specifically, we find that maximizing the sum of task reward and/or intrinsic reward based on a state entropy estimate can accelerate reinforcement learning when the distribution of high-value states and/or low-value states is uniform. we leverage this idea to train agents that maximize the sum of both task reward and/or intrinsic reward by selectively exploring high-value states and/or low-value states that are near the distribution of high-value states. we present a novel technique, value-conditional state entropy exploration (vvse), which trains agents that maximize the sum of both task reward and/or intrinsic reward by selectively exploring high-value states and/or low-value states that are near the distribution of high-value states. we show that vce accelerates reinforcement learning when the distribution of high-value states and/or low-value states is uniform.

by the fine-tuned $\text{PEGASUS}_{\text{LARGE}}$. In this case, the generated text has less repetition, but the two sentences marked in blue that are repeated don't make any sense. Furthermore, the generated text doesn't mention the Markov decision process, colored in violet in the original text, that is mentioned and used to solve the problem discussed in the original abstract.

A.5 T5-small

In Table A7, generated abstract from the test data of the RL dataset with the highest ROUGE1-F1-, ROUGE2-F1- and ROUGEL-F1-scores by the fine-tuned T5_{SMALL} is seen. T5_{SMALL} is fine-tuned with the training data of the RL dataset. The generated text is shorter than the original and contains two sentences colored in blue that seem to state the same thing in different orders of words.

Table A6: Generated abstract from the RL-test dataset by the fine-tuned $\texttt{PEGASUS}_{\texttt{LARGE}}$ on RL-dataset.

ROUGE1-F1	0.531
ROUGE2-F1	0.178
ROUGEL-F1	0.214
Original	learning a disentangled representation of the latent space has become one of the most fundamental problems studied in computer vision. recently, many generative adversarial networks (gans) have shown promising results in generating high fidelity images. however, studies to understand the semantic layout of the latent space of pre-trained models are still limited. several works train conditional gans to generate faces with required semantic attributes. unfortunately, in these attempts, the generated output is often not as photo-realistic as the unconditional state-of-the-art models. besides, they also require large computational resources and specific datasets to generate high fidelity images. in our work, we have formulated a markov decision process (mdp) over the latent space of a pre-trained gan model to learn a conditional policy for semantic manipulation along specific attributes under defined identity bounds. further, we have defined a semantic age manipulation scheme using a locally linear approximation over the latent space. results show that our learned policy samples high fidelity images with required age alterations, while preserving the identity of the person.
Generated	the task of performing age-specific semantic attribute manipulation in human face images has multiple applications. for example, face aging has been used for cross face verification and even in forensic art. the outputs are low in resolution or not comparable to the images generated by the state-of-the-art gans like progressive gan and style gan. this often limits the application of such models in downstream tasks, which require high-resolution images with particular facial attributes. a custom generative model can generate high-resolution images with required facial attributes, but training it from scratch is an arduous task. recent studies have tried to understand and utilize the latent structure of generative models. the authors of this paper have shown that a non-linear trajectory learned by the rl policy, over the latent space of a pre-trained progressive gan, performs the required age manipulation. that the vector arithmetic over the latent space has a direct association with the semantic changes over the generated vectors. that the vector arithmetic over the latent space has a direct association with the semantic changes over the generated vectors. the authors of interface gan [] have shown promising results in generating semantically rich high resolution images by traversing the latent space of a pre-trained progressive gan.

Table A7: Generated abstract from the RL-test dataset with the highest ROUGE1-F1-, ROUGE2-F1- and ROUGEL-F1-scores by the fine-tuned $\texttt{T5}_{\texttt{SMALL}}$.

ROUGE1-F1	0.686
ROUGE2-F1	0.520
ROUGEL-F1	0.544
Original	traffic congestion is a serious problem in urban areas. dynamic congestion pricing is one of the useful schemes to eliminate traffic congestion in strategic scale. however, in the reality, an optimal dynamic congestion pricing is very difficult or impossible to determine theoretically, because road networks are usually large and complicated, and behavior of road users is uncertain. to account for this challenge, this work proposes a dynamic congestion pricing method using deep reinforcement learning (drl). it is designed to eliminate traffic congestion based on observable data in general large-scale road networks, by leveraging the data-driven nature of deep reinforcement learning. one of the novel elements of the proposed method is the distributed and cooperative learning scheme. specifically, the drl is implemented by a spatial-temporally distributed manner, and cooperation among drl agents is established by novel techniques we call spatially shared reward and temporally switching learning. it enables fast and computationally efficient learning in large-scale networks. the numerical experiments using sioux falls network showed that the proposed method works well thanks to the novel learning scheme.
Generated	dynamic congestion pricing is a serious problem in urban areas. in this study, we propose a dynamic congestion pricing method using deep reinforcement learning (drl) to eliminate traffic congestion in strategic scale. in the reality, an optimal dynamic congestion pricing is very difficult or impossible to determine theoretically, because road networks are usually large and complicated, and behavior of road users is uncertain. in this study, we propose a novel distributed and cooperative learning scheme using deep reinforcement learning (drl) to eliminate congestion in large-scale networks. in the present study, the proposed method is based on spatially distributed reward and temporally switching learning.