SOLVING MATHS WORD PROBLEMS THROUGH AI

A Project Report Submitted

In Partial Fulfillment for The Degree Of

BACHELORS OF TECHNOLOGY

In

COMPUTER ENGINEERING

by

MAUWAZ AHMED FAROOQUI (18BCS048)

DEBAL HUSAIN ABBAS (18BCS046)

Under the Supervision of

DR. WASEEM AHMED

DEPARTMENT OF COMPUTER ENGINEERING

FACULTY OF ENGINEERING AND TECHNOLOGY

JAMIA MILLIA ISLAMIA, NEW DELHI – 110025

JUNE, 2022

CERTIFICATE

This is to certify that the project report entitled **Solving Maths Word Problems Through AI** submitted by **Mauwaz Ahmed Farooqui and Debal Husain Abbas** to the Department of Computer Engineering, Faculty of Engineering & Technology, Jamia Millia Islamia, New Delhi – 110025 in partial fulfillment for the award of the degree of **B. Tech in (Computer Engineering)** is a *bona fide* record of project work carried out by him under my supervision. The contents of this report, in full or in parts, have not been submitted to any other Institution or University for the award of any degree.

Dr. Waseem Ahmed
Supervisor
Department of Computer Engineering
Jamia Millia Islamia, New Delhi

New Delhi Counter signature of HOD with seal

June, 2022

DECLARATION

We declare that this project report titled **Solving Maths Word Problems Through AI** submitted in partial fulfillment of the degree of **B. Tech in (Computer Engineering)** is a record of original work carried out by us under the supervision of **Dr. Waseem Ahmed,** and has not formed the basis for the award of any other degree, in this or any other Institution or University. In keeping with the ethical practice in reporting scientific information, due acknowledgements have been made wherever the findings of others have been cited.

Debal Husain Abbas
18BCS046

Mauwaz Ahmed Farooqui
18BCS048

New Delhi-110025

18th June, 2022

ACKNOWLEDGEMENTS

In the present world of competition there is a race of existence in which those who have the will to come forward succeed. Project is like a bridge between theoretical and practical knowledge. With this willing we joined this particular project. First of all, we would like to thank the supreme power the Almighty Allah who is obviously the one who has always guided us to work on the right path of life. Without his grace this project could not become a reality. Next to him are our parents, whom we are greatly indebted to for bringing us with love and encouragement to this stage. We are highly obliged to take the opportunity to express our gratitude and sincerely thank **Dr. Waseem Ahmed (Assistant Professor, D/o Computer Engg., Jamia Millia Islamia)** for his guidance and supervision. A special thanks to our visionary **HoD, Prof. Bashir Alam** for keeping a great balance between theoretical learning and practical knowledge throughout the course**.** Moreover, we are highly obliged in taking the opportunity to sincerely thank all the staff members of the Computer Engineering. department for their generous attitude and friendly behaviour. And last but not the least we are thankful to all our teachers and friends who have always been helpful and encouraging throughout the year. A sincere thanks to all.

DEBAL HUSAIN ABBAS MAUWAZ AHMED FAROOQUI

ABSTRACT

State-of-the-art language models can match human performance on many tasks, but they still struggle to robustly perform multi-step mathematical reasoning. We find that even the largest transformer models fail to achieve high test performance, despite the conceptual simplicity of this problem distribution. Large language models like GPT-3 despite having impressive functionalities struggle to perform tasks requiring multi-step reasoning.

In this project, we propose a different methodology to minimize the errors produced in logic while solving multi-step reasoning problems. We then compare our results with the existing methodologies used in research in the domain.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

In recent years, large language models have demonstrated impressive skills across many diverse tasks. However, even the largest models falter when required to perform multi-step mathematical reasoning. Model samples frequently contain catastrophic mistakes, even after the model has been appropriately fine-tuned. Mathematical reasoning thus reveals a critical weakness in modern language models.

One significant challenge in mathematical reasoning is the high sensitivity to individual mistakes. When generating a solution, autoregressive models have no mechanism to correct their own errors. Solutions that veer off-course quickly become unrecoverable. The research paper proposes training verifiers to evaluate the correctness of model generated solutions. At test time, a fixed number of candidate solutions are sampled and the solution ranked highest by the verifier is selected. Verifiers benefit both from their inherent optionality and from verification being a simpler task than generation in general [1] [7] [9].

We propose a different methodology in solving grade school Maths word problems by using the existing GSM8K dataset. We divide the Maths word problems into sub-questions and solve these questions sequentially with the answers of previously solved sub-questions acting as contexts and the final answer being the solution of the given word problem.

We use BART encoder-decoder transformer model for solving the word problems within the limits of memory and computation available with us. We then compare our results with that mentioned in the research paper. We also try to compare our results with the methodologies proposed in recently published research papers.

CHAPTER 2

LITERATURE

2.1. About The Research

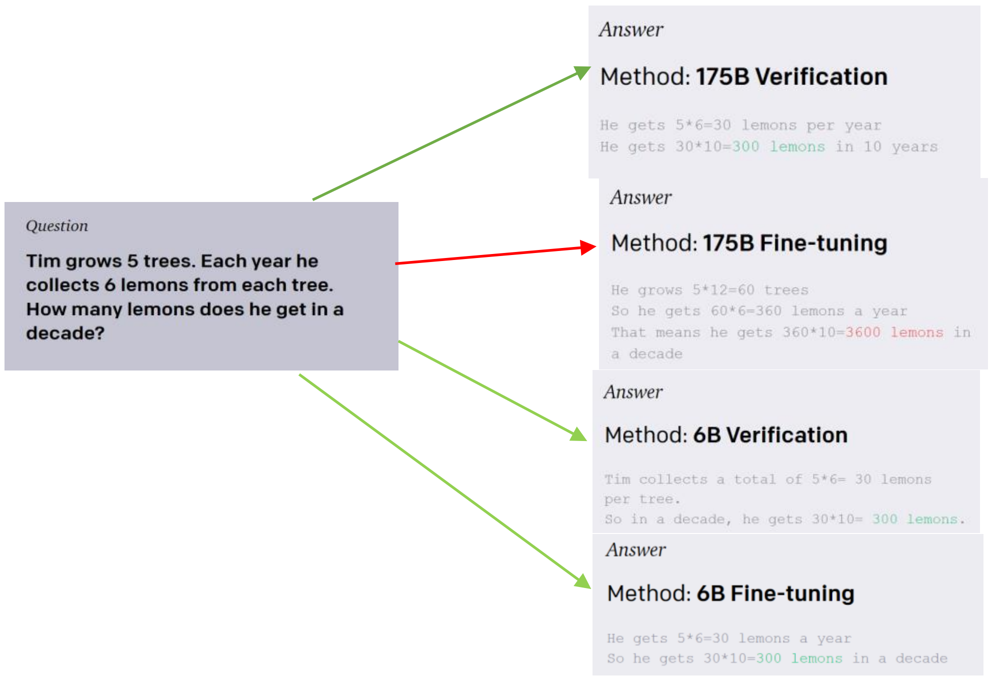
The following is an extract from the citation of the research paper used as the basis for the project [1].

```
@article{cobbe2021gsm8k,


    title = "Training Verifiers to Solve Math Word Problems”,

    author = "Cobbe, Karl and Kosaraju, Vineet and Bavarian, Mohammad
    and Chen, Mark and Jun, Heewoo and Kaiser, Lukasz and Plappert,
    Matthias and Tworek, Jerry and Hilton, Jacob and Nakano, Reiichiro
    and Hesse, Christopher and Schulman, John”,

    year = "2021",

    publisher = "Association for Computational Linguistics",

}
```

Large language models like GPT-3 have many impressive skills, including their ability to imitate many writing styles, and their extensive factual knowledge. However, they struggle to perform tasks that require accurate multistep reasoning, like solving grade school math word problems. Although the model can mimic the cadence of correct solutions, it regularly produces critical errors in logic [2].

To match human performance in complex logical domains, our models must learn to recognize their mistakes and to choose their steps carefully. To that end, we train verifiers to evaluate whether or not a proposed solution is correct. To solve a new problem, we use verifiers to select the best among many proposed solutions. We collected the new GSM8K dataset to evaluate our methods, and we are releasing this dataset to facilitate research [3].

The example shown in the Figure 2.1 below shows solutions generated by the methodology used in the research paper - verification, and the baseline method of fine-tuning.


Question

Tim grows 5 trees. Each year he collects 6 lemons from each tree. How many lemons does he get in a decade?

Answer

Method: **175B Verification**

```
He gets 5*6=30 lemons per year
He gets 30*10=300 lemons in 10 years
```

Answer

Method: **175B Fine-tuning**

```
He grows 5*12=60 trees
So he gets 60*6=360 lemons a year
That means he gets 360*10=3600 lemons in
a decade
```

Answer

Method: **6B Verification**

```
Tim collects a total of 5*6= 30 lemons
per tree.
So in a decade, he gets 30*10= 300 lemons.
```

Answer

Method: **6B Fine-tuning**

```
He gets 5*6=30 lemons a year
So he gets 30*10=300 lemons in a decade
```

Figure 2.1: Solution Generated by OpenAI GPT-3

One significant challenge in mathematical reasoning is the high sensitivity to individual mistakes. Autoregressive models, which generate each solution token by token, have no mechanism to correct their own errors. Solutions that veer off-course quickly become unrecoverable, as can be seen in the examples provided [6].

We address this problem by training verifiers to evaluate the correctness of model-generated solutions. Verifiers are given many possible solutions, all written by the model itself, and they are trained to decide which ones, if any, are correct.

To solve a new problem at test time, we generate 100 candidate solutions and then select the solution that is ranked highest by the verifier. Verifiers benefit from this inherent optionality, as well as from the fact that verification is often a simpler task than generation.

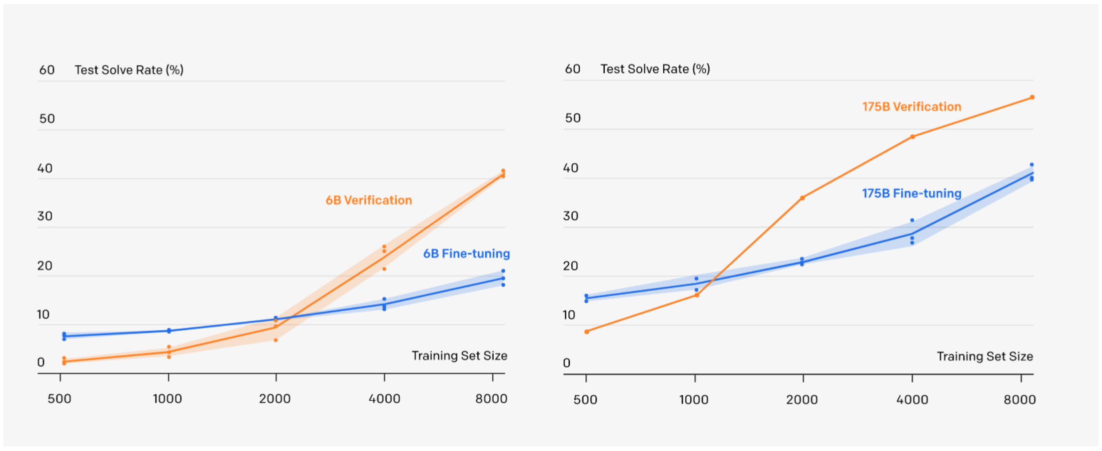


Figure 2.2: Evaluating OpenAI GPT-3 model on different Parameters

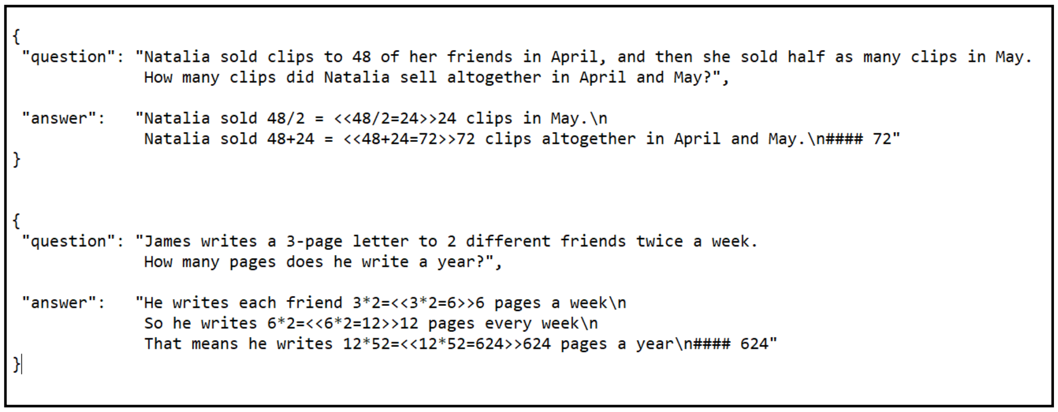
As evident from Figure 2.2, we find that we get a strong boost in performance from verification, as long as the dataset is large enough. With datasets that are too small, we believe that the verifiers overfit by memorizing the final answers in the training set, rather than learning any more useful properties of mathematical reasoning [2].

On the full training set, 6B parameter verification slightly outperforms a fine-tuned 175B parameter model, giving a performance boost that is approximately equivalent to a 30x model size increase. Moreover, verification appears to scale more effectively with additional data, if we extrapolate based on current results [9].

2.2. Dataset Used for Training and Testing the Model

GSM8K consists of 8.5K high quality grade school math word problems created by human problem writers [3]. Each problem takes between 2 and 8 steps to solve, and solutions primarily involve performing a sequence of elementary calculations using basic arithmetic operations ($+ - \times \div$) to reach the final answer. We segmented these into 7,500 training problems and 1,000 test problems.

Solutions in GSM8K are written as natural language rather than as pure math expressions as shown in Figure 2.3. By sticking to natural language, model-generated solutions are more readily interpretable by humans, and our methods remain relatively domain agnostic.



```
{
 "question": "Natalia sold clips to 48 of her friends in April, and then she sold half as many clips in May.
             How many clips did Natalia sell altogether in April and May?",

 "answer":   "Natalia sold 48/2 = <<48/2=24>>24 clips in May.\n
             Natalia sold 48+24 = <<48+24=72>>72 clips altogether in April and May.\n#### 72"
}


{
 "question": "James writes a 3-page letter to 2 different friends twice a week.
             How many pages does he write a year?",

 "answer":   "He writes each friend 3*2=<<3*2=6>>6 pages a week\n
             So he writes 6*2=<<6*2=12>>12 pages every week\n
             That means he writes 12*52=<<12*52=624>>624 pages a year\n#### 624"
}
```

Figure 2.3: Dataset Format – JSON file

Table 2.1: Number of Word Problems in the Dataset

	Number of Word Problems	
	SubTask 1	SubTask 2
Train.csv	6552	9306
Dev.csv	740	1046
Test.csv	100	100

The proposed methodology discusses the division of the problem statement into two subtasks. Table 2.1 shows the split of dataset into Train, Test and Dev set for the two subtasks.

This is the dataset used in the research paper mentioned above. The dataset used by us is a derived form of the existing GSM8K dataset used in the research paper. We have discussed the need and process to derive the dataset for our methodology in Chapter 4.

CHAPTER 3

PRELIMINARIES

3.1. About Deep Learning

Deep learning is a machine learning technique that teaches computers to do what comes naturally to humans: learn by example. Deep learning is a key technology behind driverless cars, enabling them to recognize a stop sign, or to distinguish a pedestrian from a lamppost. It is the key to voice control in consumer devices like phones, tablets, TVs, and hands-free speakers. Difference between Artificial Intelligence, Machine Learning and Deep Learning is shown in Figure 3.1.

In deep learning, a computer model learns to perform classification tasks directly from images, text, or sound. Deep learning models can achieve state-of-the-art accuracy, sometimes exceeding human-level performance [21]. Models are trained by using a large set of labeled data and neural network architectures that contain many layers.

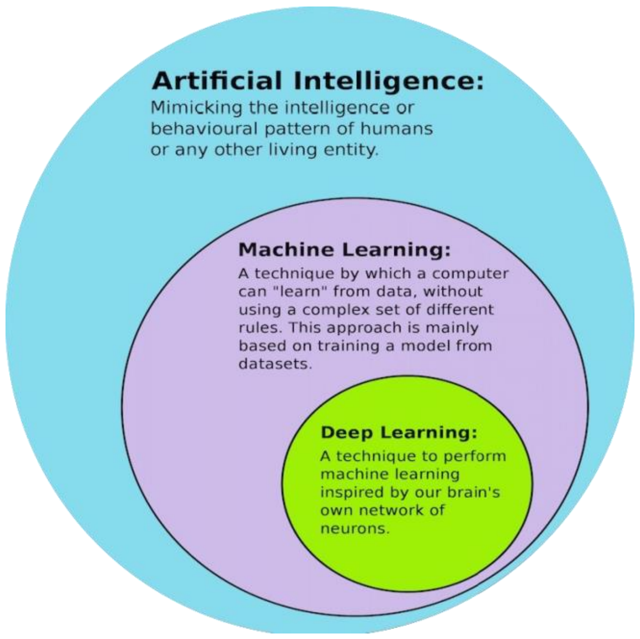

Artificial Intelligence:
Mimicking the intelligence or behavioural pattern of humans or any other living entity.
Machine Learning:
A technique by which a computer can "learn" from data, without using a complex set of different rules. This approach is mainly based on training a model from datasets.
Deep Learning:
A technique to perform machine learning inspired by our brain's own network of neurons.

Figure 3.1: Artificial Intelligence vs. Machine Learning vs. Deep Learning

3.2. Transfer Learning

Transfer learning generally refers to a process where a model trained on one problem is used in some way on a second related problem. In deep learning, transfer learning is a technique whereby a neural network model is first trained on a problem similar to the problem that is being solved. One or more layers from the trained model are then used in a new model trained on the problem of interest. Multiple deep learning domains use this approach, including Image Classification, Natural Language Processing, and even Gaming! The ability to adapt a trained model to another task is incredibly valuable. This is typically understood in a supervised learning context, where the input is the same but the target may be of a different nature. For example, we may learn about one set of visual categories, such as cats and dogs, in the first setting, then learn about a different set of visual categories, such as ants and wasps, in the second setting.

Transfer learning has the benefit of decreasing the training time for a neural network model and can result in lower generalization error. The weights in re-used layers may be used as the starting point for the training process and adapted in response to the new problem. This usage treats transfer learning as a type of weight initialization scheme. This may be useful when the first related problem has a lot more labeled data than the problem of interest and the similarity in the structure of the problem may be useful in both contexts [23].

3.2.1. Transfer Learning for Natural Language Processing

The word embeddings used as major preprocessing step in all text-based applications are a more informative way of representing words, compared to one-hot encodings. They are widely used, and different variants exist. Typically, these variants differ in the corpus they originate from, such as Wikipedia, news articles, etc., and the differences in the embedding models. It is important to understand the background of these models and corpuses in order to know whether transfer learning with word embeddings is sensible. Essentially, using word embeddings means that you are using a featuriser or the embedding network to convert words to vectors.

An alternative to the standard pre-trained word embeddings is to fine-tune the embeddings on a large unsupervised set of documents. Note that this is only an option if a large set of documents is available. This means that if you have a large corpus on competition law, you could train the word embeddings for the domain-specific words, by starting from pre-trained word embeddings for the other, more general words. Typically, starting for pre-trained word embeddings will speed up the whole process, and will make training your own word embeddings easier. Note that it is still harder using word embeddings out-of-the-box and requires some knowledge on how to prepare the corpus for word embedding training [28].

Gensim, spacy and FastText are three great frameworks that allow you to quickly use word embeddings in your machine learning application. In addition, they also support the training of custom word embeddings.

3.4. Attention Models

A neural network is considered to be an effort to mimic human brain actions in a simplified manner. Attention Mechanism is also an attempt to implement the same action of selectively concentrating on a few relevant things, while ignoring others in deep neural networks. The attention mechanism emerged as an improvement over the encoder decoder-based neural machine translation system in natural language processing (NLP). Later, this mechanism, or its variants, was used in other applications, including computer vision, speech processing, etc. [25].

Before Bahdanau et al proposed the first Attention model in 2015, neural machine translation was based on encoder-decoder RNNs/LSTMs. Both encoder and decoder are stacks of LSTM/RNN units. In short, there are two RNNs/LSTMs. One we call the encoder – this reads the input sentence and tries to make sense of it, before summarizing it. It passes the summary (context vector) to the decoder which translates the input sentence by just seeing it.

The main drawback of this approach is evident. If the encoder makes a bad summary, the translation will also be bad. And indeed it has been observed that the encoder creates a bad summary when it tries to understand longer sentences. It is called the *long-range dependency problem of RNN/LSTMs*.

RNNs cannot remember longer sentences and sequences due to the vanishing/exploding gradient problem. It can remember the parts which it has just seen. Even *Cho et al,* who proposed the encoder-decoder network, demonstrated that the performance of the encoder-decoder network degrades rapidly as the length of the input sentence increases [14].

Although an LSTM is supposed to capture the long-range dependency better than the RNN, it tends to become forgetful in specific cases. Another problem is that there is no way to give more importance to some of the input words compared to others while translating the sentence.

Now, let's say, we want to predict the next word in a sentence, and its context is located a few words back. Here's an example – *"Despite originally being from Uttar Pradesh, as he was brought up in Bengal, he is more comfortable in Bengali"*. In these groups of sentences, if we want to predict the word *"Bengali"*, the phrase *"brought up"* and *"Bengal"*- these two should be given more weight while predicting it. And although *Uttar Pradesh* is another state's name, it should be "ignored".

So is there any way we can keep all the relevant information in the input sentences intact while creating the context vector?

Bahdanau et al came up with a simple but elegant idea where they suggested that not only can all the input words be taken into account in the context vector, but relative importance should also be given to each one of them [29].

So, whenever the proposed model generates a sentence, it searches for a set of positions in the encoder hidden states where the most relevant information is available. This idea is called 'Attention'.

3.5. Transformer Models

The Transformer architecture follows an encoder-decoder structure, but does not rely on recurrence and convolutions in order to generate an output.

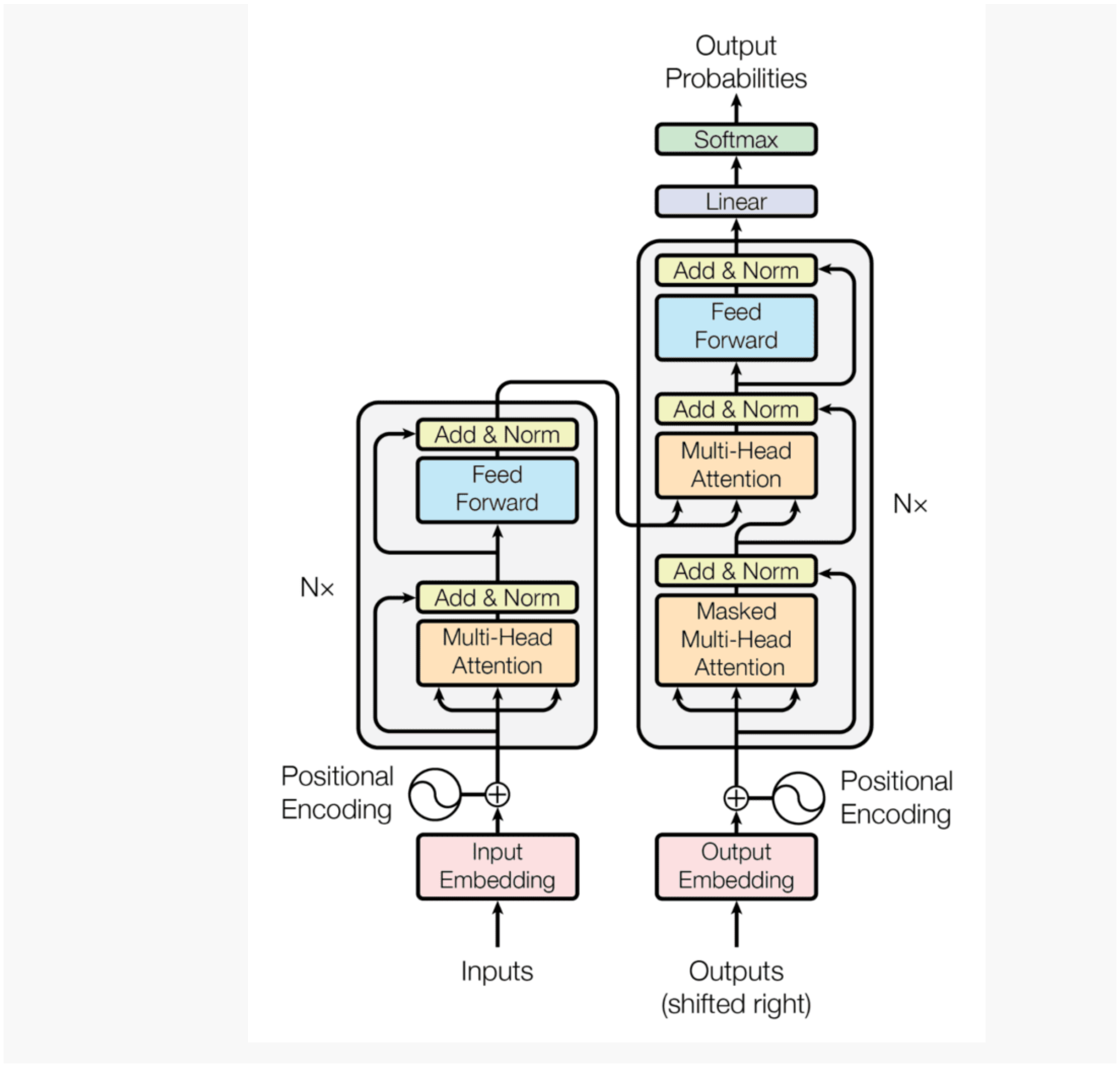


Figure 3.2: The Encoder-Decoder Structure of the Transformer Architecture

In a nutshell, the task of the encoder, on the left half of the Transformer architecture, is to map an input sequence to a sequence of continuous representations, which is then fed into a decoder as shown in Figure 3.2.

The decoder, on the right half of the architecture, receives the output of the encoder together with the decoder output at the previous time step, to generate an output sequence [27].

The Transformer model runs as follows:

1. Each word forming an input sequence is transformed into a dmodel-dimensional embedding vector.
2. Each embedding vector representing an input word is augmented by summing it (element-wise) to a positional encoding vector of the same dmodel length, hence introducing positional information into the input.
3. The augmented embedding vectors are fed into the encoder block, consisting of the two sublayers explained above. Since the encoder attends to all words in the input sequence, irrespective if they precede or succeed the word under consideration, then the Transformer encoder is *bidirectional*.
4. The decoder receives as input its own predicted output word at time-step, t–1.
5. The input to the decoder is also augmented by positional encoding, in the same manner as this is done on the encoder side.
6. The augmented decoder input is fed into the three sublayers comprising the decoder block explained above. Masking is applied in the first sublayer, in order to stop the decoder from attending to succeeding words. At the second sublayer, the decoder also receives the output of the encoder, which now allows the decoder to attend to all of the words in the input sequence.
7. The output of the decoder finally passes through a fully connected layer, followed by a softmax layer, to generate a prediction for the next word of the output sequence.

3.5.1. Encoder Models - BERT

Each encoder consists of two major components: a self-attention mechanism and a feed-forward neural network. The self-attention mechanism accepts input encodings from the previous encoder and weighs their relevance to each other to generate output encodings. The feed-forward neural network further processes each output encoding individually. These output encodings are then passed to the next encoder as its input, as well as to the decoders [4].

The first encoder takes positional information and embeddings of the input sequence as its input, rather than encodings. The positional information is necessary for the transformer to make use of the order of the sequence, because no other part of the transformer makes use of this. The encoder is bidirectional. Attention can be placed on tokens before and after the current token.

BERT stands for **B**idirectional **E**ncoder **R**epresentations from **T**ransformers. It is designed to pre-train deep bidirectional representations from unlabeled text by jointly conditioning on both left and right context. As a result, the pre-trained BERT model can be fine-tuned with just one additional output layer to create state-of-the-art models for a wide range of NLP tasks [12]. The major features of BERT are as follows:

- BERT is based on the Transformer architecture.
- BERT is pre-trained on a large corpus of unlabelled text including the entire Wikipedia (that's 2,500 million words!) and Book Corpus (800 million words). This **pre-training** step is half the magic behind BERT's success. This is because as we train a model on a large text corpus, our model starts to pick up the deeper and intimate understandings of how the language works. This knowledge is the **swiss army knife** that is useful for almost any NLP task.
- BERT is a **"deeply bidirectional"** model. Bidirectional means that BERT learns information from both the left and the right side of a token's context during the training phase.
- The bidirectionality of a model is important for truly understanding the meaning of a language.

Let's see an example to illustrate this. There are two sentences in this example and both of them involve the word "bank":

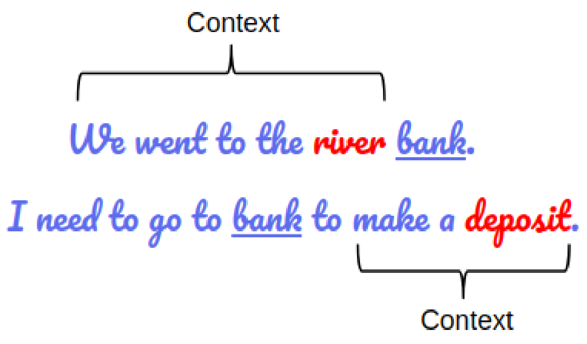

Context
We went to the river bank.
I need to go to bank to make a deposit.
Context

Figure 3.3: BERT captures both the left and right context

As shown in Figure 3.3, if we try to predict the nature of the word "bank" by only taking either the left or the right context, then we will be making an error in at least one of the two given examples.

One way to deal with this is to consider both the left and the right context before making a prediction. That's exactly what BERT does! The most impressive aspect of BERT is that we can fine-tune it by adding just a couple of additional output layers to create state-of-the-art models for a variety of NLP tasks [16]. So, the new approach to solving NLP tasks became a 2-step process:

1. Train a language model on a large unlabelled text corpus (unsupervised or semi-supervised)
2. Fine-tune this large model to specific NLP tasks to utilize the large repository of knowledge this model has gained (supervised)

With that context, let's understand how BERT takes over from here to build a model that will become a benchmark of excellence in NLP for a long time [25].

3.5.1.1. BERT’s Architecture

The BERT architecture builds on top of Transformer [7]. We currently have two variants available:

- *BERT Base*: 12 layers (transformer blocks), 12 attention heads, and 110 million parameters
- *BERT Large*: 24 layers (transformer blocks), 16 attention heads and, 340 million parameters

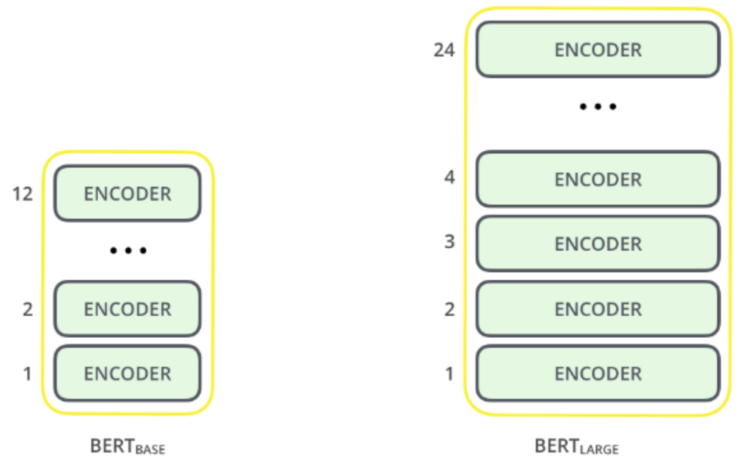


Figure 3.4: BERT Architecture

The BERT Base architecture has the same model size as OpenAI’s GPT for comparison purposes. All of these Transformer layers are **Encoder**-only blocks as shown in Figure 3.4.

3.5.1.2. Text Preprocessing

The developers behind BERT have added a specific set of rules to represent the input text for the model. Many of these are creative design choices that make the model even better. The word embedding process is shown in Figure 3.5.

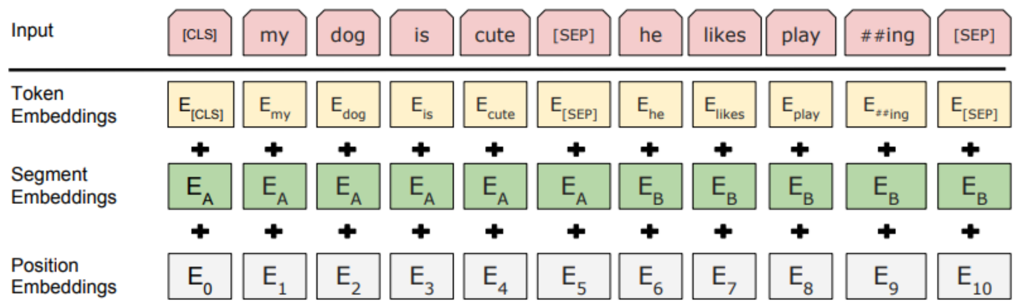

Input
[CLS]
my
dog
is
cute
[SEP]
he
likes
play
##ing
[SEP]
Token Embeddings
$E_{[CLS]}$ E_{my} E_{dog} E_{is} E_{cute} $E_{[SEP]}$ E_{he} E_{likes} E_{play} $E_{\#\#ing}$ $E_{[SEP]}$
Segment Embeddings
E_A E_A E_A E_A E_A E_A E_B E_B E_B E_B E_B
Position Embeddings
E_0 E_1 E_2 E_3 E_4 E_5 E_6 E_7 E_8 E_9 E_{10}

Figure 3.5: Word Embedding

For starters, every input embedding is a combination of 3 embeddings:

1. Position Embeddings: BERT learns and uses positional embeddings to express the position of words in a sentence. These are added to overcome the limitation of Transformer which, unlike an RNN, is not able to capture "sequence" or "order" information
2. Segment Embeddings: BERT can also take sentence pairs as inputs for tasks (Question-Answering). That's why it learns a unique embedding for the first and the second sentences to help the model distinguish between them. In the above example, all the tokens marked as EA belong to sentence A (and similarly for EB)
3. Token Embeddings: These are the embeddings learned for the specific token from the WordPiece token vocabulary

For a given token, its input representation is constructed by summing the corresponding token, segment, and position embeddings. Such a comprehensive embedding scheme contains a lot of useful information for the model.

These combinations of preprocessing steps make BERT so versatile. This implies that without making any major change in the model's architecture, we can easily train it on multiple kinds of NLP tasks [12].

BERT is pre-trained on two NLP tasks:

- Masked Language Modeling
- Next Sentence Prediction

3.5.2. Decoder Models – GPT-2

Each decoder consists of three major components: a self-attention mechanism, an attention mechanism over the encodings, and a feed-forward neural network. The decoder functions in a similar fashion to the encoder, but an additional attention mechanism is inserted which instead draws relevant information from the encodings generated by the encoders. This mechanism can also be called the *encoder-decoder attention*.

Like the first encoder, the first decoder takes positional information and embeddings of the output sequence as its input, rather than encodings. The transformer must not use the current or future output to predict an output, so the output sequence must be partially masked to prevent this reverse information flow. This allows for autoregressive text generation. For all attention heads, attention can't be placed on following tokens. The last decoder is followed by a final linear transformation and softmax layer, to produce the output probabilities over the vocabulary [5].

GPT has a decoder-only architecture. **GPT-2** or **Generative Pre-Trained Transformer 2**, is an unsupervised transformer language model. The corpus it was trained on, called WebText, contains slightly over 8 million documents for a total of 40 GB of text from URLs shared in Reddit submissions with at least 3 upvotes.

GPT-2 is a transformers model pretrained on a very large corpus of English data in a self-supervised fashion. This means it was pretrained on the raw texts only, with no humans labelling them in any way (which is why it can use lots of publicly available data) with an automatic process to generate inputs and labels from those texts. More precisely, it was trained to guess the next word in sentences [18].

More precisely, inputs are sequences of continuous text of a certain length and the targets are the same sequence, shifted one token (word or piece of word) to the right. The model uses internally a mask-mechanism to make sure the predictions for the token i only uses the inputs from 1 to ‘i’ but not the future tokens [5].

This way, the model learns an inner representation of the English language that can then be used to extract features useful for downstream tasks. The model is best at what it was pretrained for however, which is generating texts from a prompt.

3.5.2.1. Pre-Processing

The texts are tokenized using a byte-level version of Byte Pair Encoding (BPE) (for unicode characters) and a vocabulary size of 50,257. The inputs are sequences of 1024 consecutive tokens. The larger model was trained on 256 cloud TPU v3 cores [18].

3.5.2.2. Architecture

GPT's architecture itself was a twelve-layer decoder-only transformer, using twelve masked self-attention heads, with 64 dimensional states each (for a total of 768). Rather than simple stochastic gradient descent, the Adam optimization algorithm was used; the learning rate was increased linearly from zero over the first 2,000 updates, to a maximum of 2.5×10^{-4}, and annealed to 0 using a cosine schedule [19].

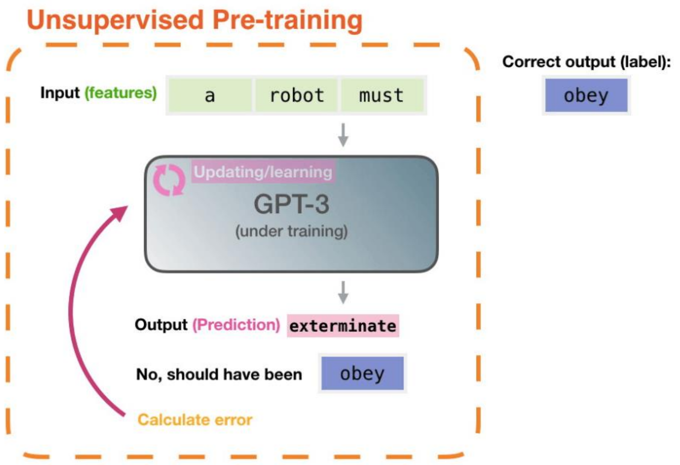


Figure 3.6: GPT-2 Architecture

GPT-2 was created as a direct scale-up of GPT, with both its parameter count and dataset size increased by a factor of 10. Both are unsupervised transformer models trained to generate text by predicting the next word in a sequence of tokens as evident from Figure 3.6. The GPT-2 model has 1.5 billion parameters, and was trained on a dataset of 8 million web pages. While GPT-2 was reinforced on very simple criteria (interpreting a sequence of words in a text sample and predicting the most likely next word), it produces full sentences and paragraphs by continuing to predict additional words, generating fully comprehensible (and semantically meaningful) statements in natural language. Notably, GPT-2 was evaluated on its performance on tasks in a zero-shot setting.

3.5.2.3. Performance

Due to the broadness of its dataset, and the broadness of its approach, GPT-2 became capable of performing a diverse range of tasks beyond simple text generation: answering questions, summarizing, and even translating between languages in a variety of specific domains, without being instructed in anything beyond how to predict the next word in a sequence.

3.5.3. Encoder-Decoder Models - BART

The original Transformer is based on an encoder-decoder architecture and is a classic sequence-to-sequence model. The model's input and output are in the form of a sequence (text), and the encoder learns a high-dimensional representation of the input, which is then mapped to the output by the decoder. This architecture introduced a new form of learning for language-related tasks and, thus, the models spawned from it achieve outstanding results overtaking the existing deep neural network-based methods.

BART is one such Transformer model that takes components from other Transformer models and improves the pretraining learning. BART or Bidirectional and Auto-Regressive [10].

Transformers was proposed in the *BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension* paper. The BART HuggingFace model allows the pre-trained weights and weights fine-tuned on question-answering, text summarization, conditional text generation, mask filling, and sequence classification.

BART is a sequence-to-sequence model trained as a denoising autoencoder. This means that a fine-tuned BART model can take a text sequence (for example, English) as input and produce a different text sequence at the output (for example, French). This type of model is relevant for machine translation, question-answering, text summarization, or sequence. Another task is sentence entailment which, given two or more sentences, evaluates whether the sentences are logical extensions or are logically related to a given statement [11].

Since the unsupervised pretraining of BART results in a language model, it can be fine-tuned to domain specific tasks.

- Bart uses a standard seq2seq/machine translation architecture with a bidirectional encoder (like BERT) and a left-to-right decoder (like GPT).
- The pretraining task involves randomly shuffling the order of the original sentences and a novel in-filling scheme, where spans of text are replaced with a single mask token.
- BART is particularly effective when fine-tuned for text generation but also works well for comprehension tasks. It matches the performance of RoBERTa with comparable training resources on GLUE and SQuAD, achieves new state-of-the-art results on a range of abstractive dialogue, question answering, and summarization tasks, with gains of up to 6 ROUGE.

Although we separate the decoder from an encoder, the input to the decoder would still be a learned representation (or embedding) of the original text sequence. Thus, BART attaches the bi-directional encoder to the autoregressive decoder to create a denoising auto-encoder architecture [10]. And based on these two components, the final BART model would look similar to as shown in Figure 3.7.

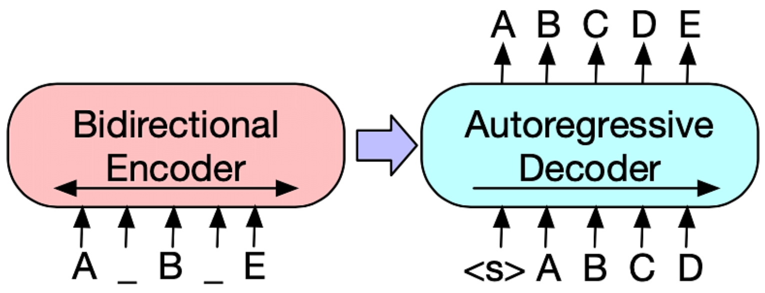


Figure 3.7: BART Architecture

In the Figure 3.7, the input sequence is a masked (or noisy) version of [ABCDE] transformed into [A[MASK]B[MASK]E]. The encoder looks at the entire sequence and learns high-dimensional representations with bi-directional information. The decoder takes these thought vectors and regressively predicts the next token. Learning occurs by computing and optimizing the negative log-likelihood as mapped with the target [ABCDE].

CHAPTER 4

METHODOLOGY

4.1. Our Proposed Method

To solve grade school Maths word problems we break down this task into the following two sub-problems –

1. <u>SubTask 1</u> – **Sub-Question Generation** - To create sub-questions from the given question
2. <u>SubTask 2</u> – **Solver** - Trying to solve each sub-question using the main question and answers of previous sub questions as the context (source of information)

The example shown in Figure 4.1 depicts the working of the above two tasks.

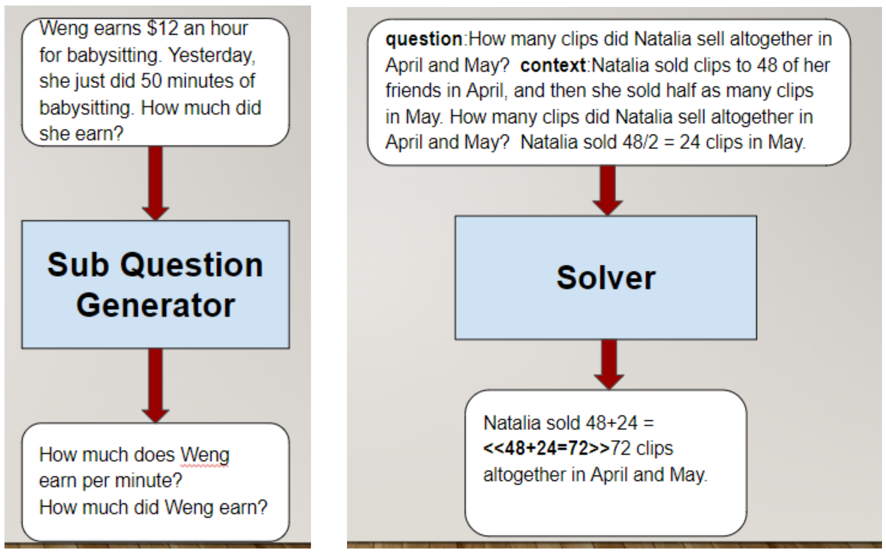

Weng earns $12 an hour for babysitting. Yesterday, she just did 50 minutes of babysitting. How much did she earn?
Sub Question Generator
How much does Weng earn per minute?
How much did Weng earn?
question:How many clips did Natalia sell altogether in April and May? **context**:Natalia sold clips to 48 of her friends in April, and then she sold half as many clips in May. How many clips did Natalia sell altogether in April and May? Natalia sold 48/2 = 24 clips in May.
Solver
Natalia sold 48+24 = **<<48+24=72>>**72 clips altogether in April and May.

Figure 4.1: Working of SubTask 1 and SubTask 2

Solving each subquestion sequentially generates the answer which is the final solution of the original word problem. At any given level, the sub-question requires two inputs – the original question and the solution to the previous question. Both these inputs act as context in solving the given sub-question at hand. Figure 4.2 depics the architecture of the proposed methodology.

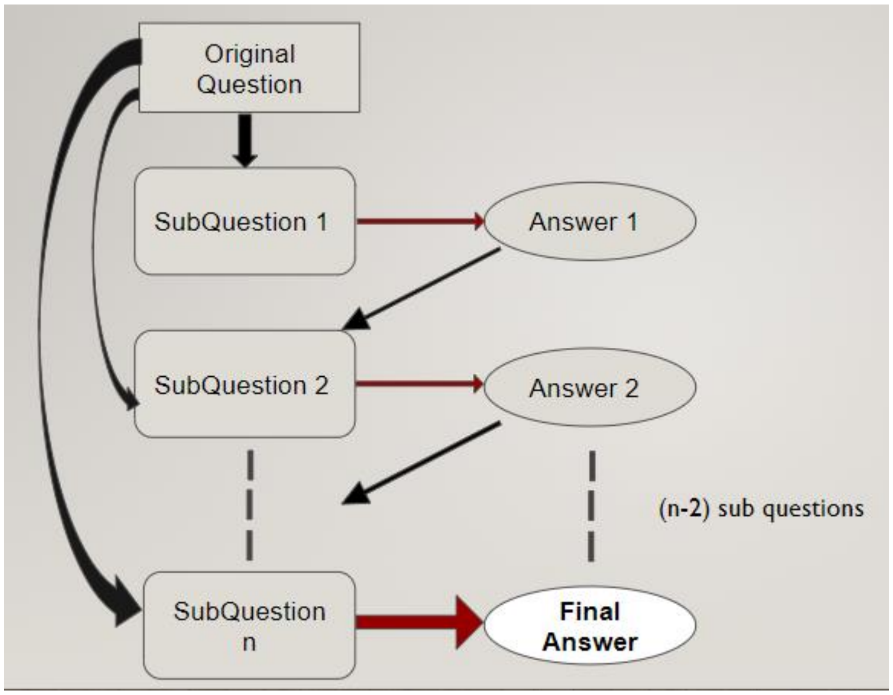


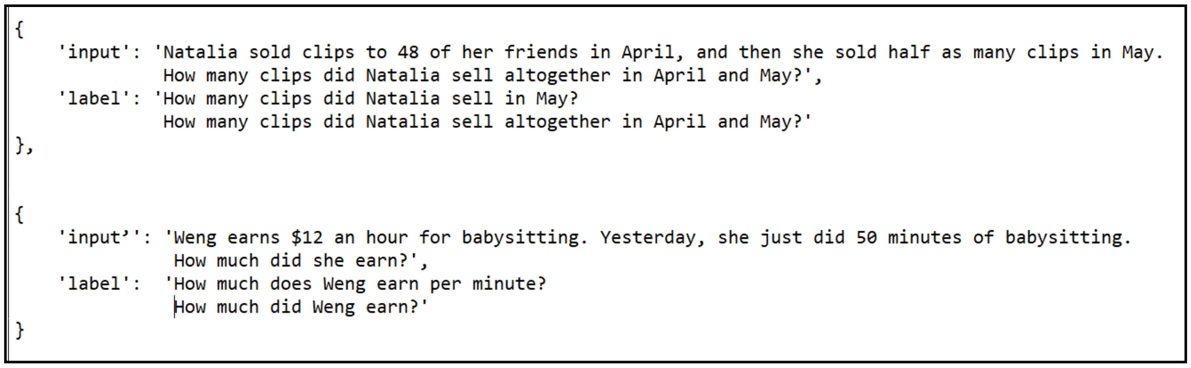
Figure 4.2: Architecture of Proposed Methodology

4.2. Dataset Preparation

The GSM8K dataset consists of ‘question’ and ‘answer’ keys in the JSON format. Since, we have broken down the problem statement into subtasks, we require dataset in a form which is suitable for each task to be performed.

4.2.1. Dataset used in SubTask 1

SubTask 1 involves breaking down of the given word-problem into sub-questions. Therefore, we need a dataset which is in the form as shown in Figure 4.3 where ‘input’ label consists of the original question and the ‘output’ label consists of the sub-questions generated from the input.



```
{
    'input': 'Natalia sold clips to 48 of her friends in April, and then she sold half as many clips in May.
              How many clips did Natalia sell altogether in April and May?',
    'label': 'How many clips did Natalia sell in May?
              How many clips did Natalia sell altogether in April and May?'
},


{
    'input’': 'Weng earns $12 an hour for babysitting. Yesterday, she just did 50 minutes of babysitting.
               How much did she earn?',
    'label':  'How much does Weng earn per minute?
               How much did Weng earn?'
}
```

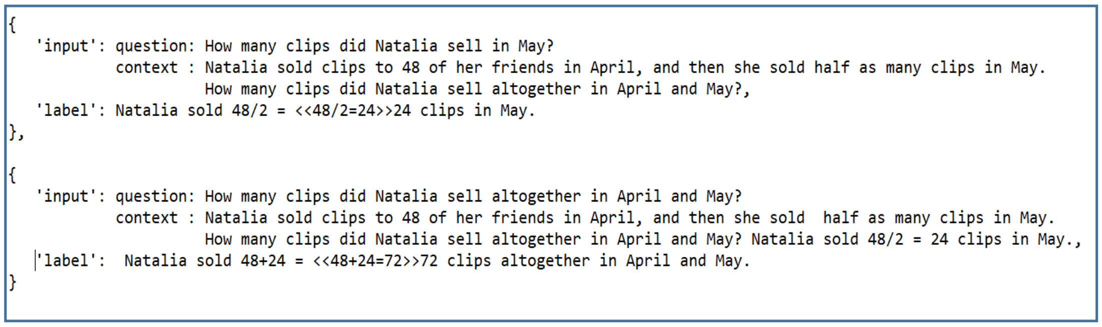
Figure 4.3: Dataset used in SubTask 1

4.2.2. Dataset used in SubTask 2

In the Dataset used for Subtask 2 (as shown in Figure 4.4), we use calculator annotations, and question-context keys. The input label is the sub-question to be solved and also includes the result from previous sub-question as context. The label consists of the final answer.

Calculator Annotations are used to take away the task of performing the arithmetic calculations from the model and perform them using a simple function defined by us. Example of such annotations ***<<48/2=24>>***.

We use Question and Context keys in our dataset as they have been used previously in similar tasks and lead to higher accuracy by aligning the texts correspondingly. This helps the model to identify what it needs to answer and where to find the information for.

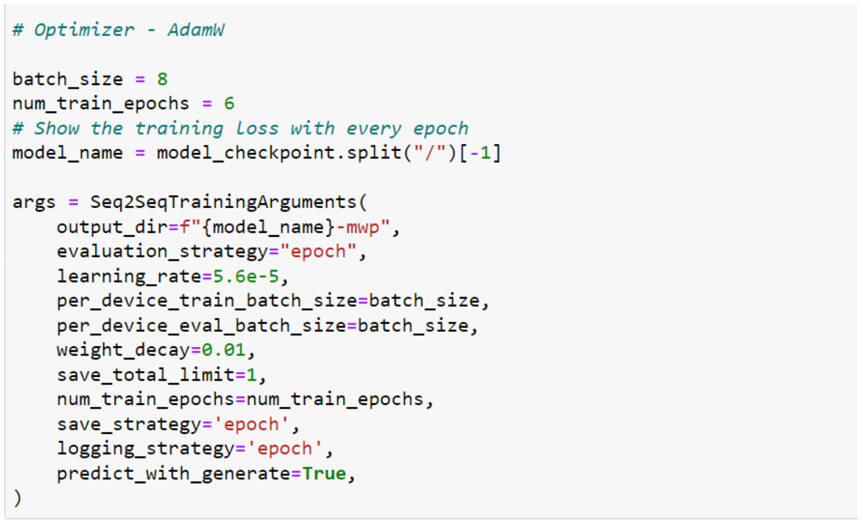


```
{
   'input': question: How many clips did Natalia sell in May?
            context : Natalia sold clips to 48 of her friends in April, and then she sold half as many clips in May.
                      How many clips did Natalia sell altogether in April and May?,
   'label': Natalia sold 48/2 = <<48/2=24>>24 clips in May.
},

{
   'input': question: How many clips did Natalia sell altogether in April and May?
            context : Natalia sold clips to 48 of her friends in April, and then she sold  half as many clips in May.
                      How many clips did Natalia sell altogether in April and May? Natalia sold 48/2 = 24 clips in May.,
   'label':  Natalia sold 48+24 = <<48+24=72>>72 clips altogether in April and May.
}
```

Figure 4.4: Dataset used in SubTask 2

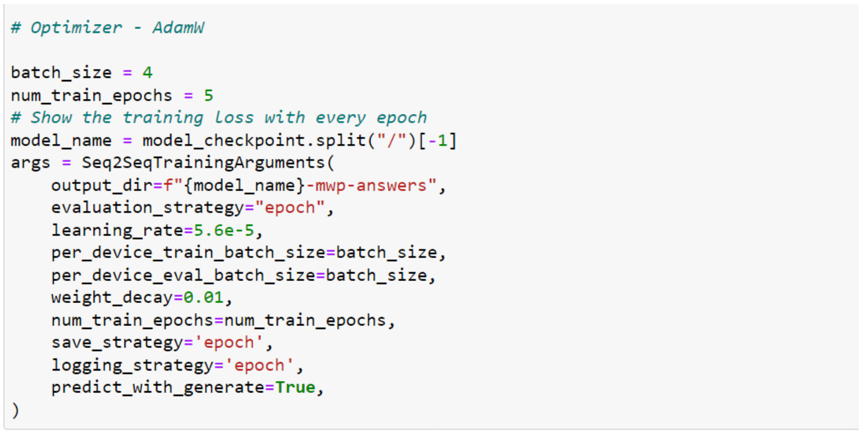
4.3. Training BART

We have used HuggingFace MBart-50 which is an Encoder-Decoder architecture based transformer model. MBart-50 is created using the original *mbart-large-cc25* checkpoint by extending its embedding layers with randomly initialized vectors for an extra set of 25 language tokens and then pre-trained on 50 languages. Figure 4.5 and Figure 4.6 mentions the training parameters used in both subtasks.



```
# Optimizer - AdamW

batch_size = 8
num_train_epochs = 6
# Show the training loss with every epoch
model_name = model_checkpoint.split("/")[-1]

args = Seq2SeqTrainingArguments(
    output_dir=f"{model_name}-mwp",
    evaluation_strategy="epoch",
    learning_rate=5.6e-5,
    per_device_train_batch_size=batch_size,
    per_device_eval_batch_size=batch_size,
    weight_decay=0.01,
    save_total_limit=1,
    num_train_epochs=num_train_epochs,
    save_strategy='epoch',
    logging_strategy='epoch',
    predict_with_generate=True,
)
```

Figure 4.5: Training Parameters for SubTask-1



```
# Optimizer - AdamW

batch_size = 4
num_train_epochs = 5
# Show the training loss with every epoch
model_name = model_checkpoint.split("/")[-1]
args = Seq2SeqTrainingArguments(
    output_dir=f"{model_name}-mwp-answers",
    evaluation_strategy="epoch",
    learning_rate=5.6e-5,
    per_device_train_batch_size=batch_size,
    per_device_eval_batch_size=batch_size,
    weight_decay=0.01,
    num_train_epochs=num_train_epochs,
    save_strategy='epoch',
    logging_strategy='epoch',
    predict_with_generate=True,
)
```

Figure 4.6: Training Parameters for SubTask-2

4.4. Evaluation

4.4.1. ROUGE Score

ROUGE-N measures the number of matching ‘n-grams’ between our model-generated text and a ‘reference’. For ROUGE-1 and ROUGE-2 we would be measuring the match-rate of unigrams and bigrams between our model output and reference [21].

F1-score gives us a reliable measure of our model performance that relies on both precision and recall. That gives us a reliable measure of our model performance that relies not only on the model capturing as many words as possible (recall) but doing so without outputting irrelevant words (precision).

ROUGE-L measures the longest common subsequence (LCS) between our model output and reference. The idea here is that a longer shared sequence would indicate more similarity between the two sequences. The representation of ROUGE score is shown in Figure 4.7.

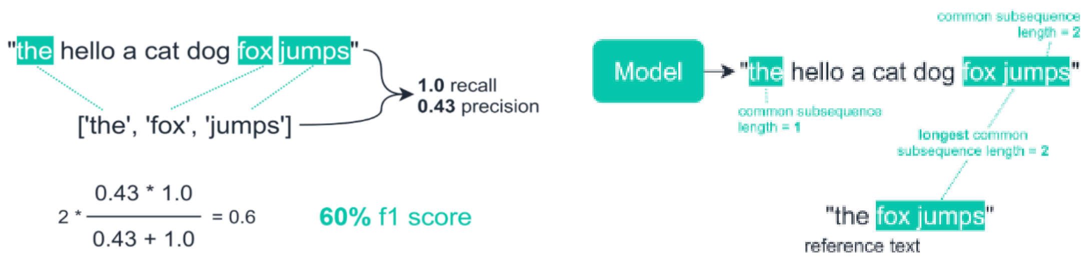
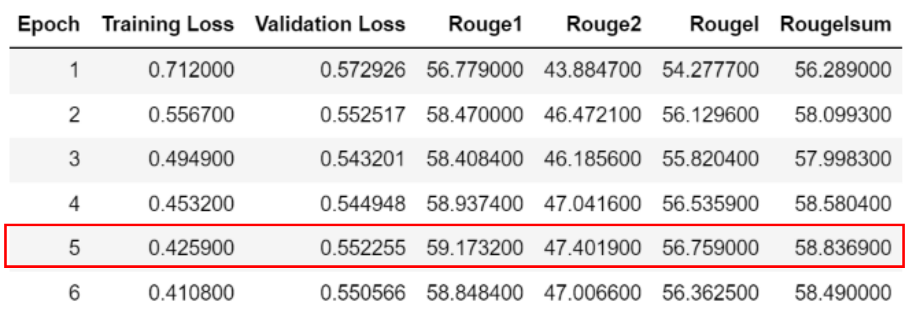

"the hello a cat dog fox jumps"
['the', 'fox', 'jumps']
1.0 recall
0.43 precision
2 * (0.43 * 1.0) / (0.43 + 1.0) = 0.6
60% f1 score
Model
"the hello a cat dog fox jumps"
common subsequence length = 2
common subsequence length = 1
longest common subsequence length = 2
"the fox jumps"
reference text

Figure 4.7: Precision, Recall, F1-Score for Rouge and Rouge-L

4.4.2. Evaluation of SubTask-1 and SubTask-2

We trained the model for Subtask-1 for 6 epochs with batch size of four. For Subtask-2, we trained our model for 5 epochs with batch size of four. Results are shown in Figure 4.7 and Figure 4.8. We have used Rogue Score as our metric to evaluate how good are the sequences generated. We have taken the checkpoint with maximum Rouge-L as our metric for choosing the best model for both the sub-tasks.



Epoch	Training Loss	Validation Loss	Rouge1	Rouge2	Rougel	Rougelsum
1	0.712000	0.572926	56.779000	43.884700	54.277700	56.289000
2	0.556700	0.552517	58.470000	46.472100	56.129600	58.099300
3	0.494900	0.543201	58.408400	46.185600	55.820400	57.998300
4	0.453200	0.544948	58.937400	47.041600	56.535900	58.580400
5	0.425900	0.552255	59.173200	47.401900	56.759000	58.836900
6	0.410800	0.550566	58.848400	47.006600	56.362500	58.490000

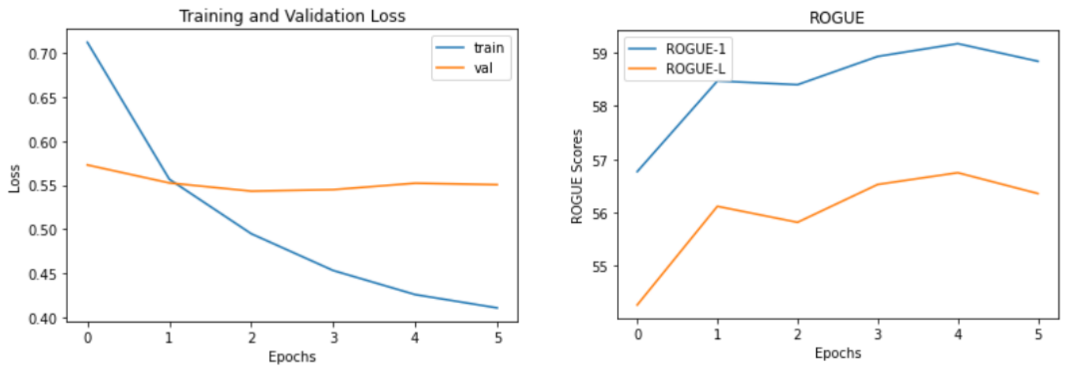
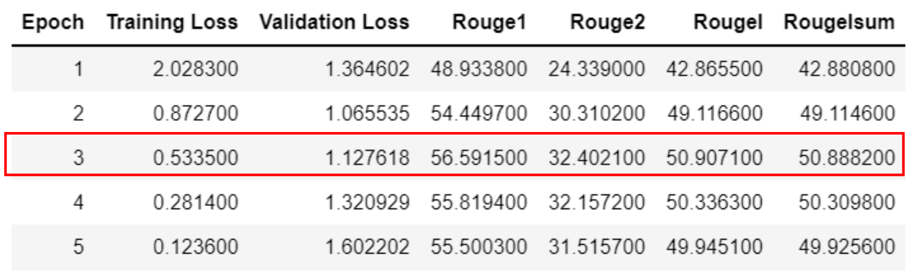


Figure 4.8: SubTask 1 Evaluation



Epoch	Training Loss	Validation Loss	Rouge1	Rouge2	Rougel	Rougelsum
1	2.028300	1.364602	48.933800	24.339000	42.865500	42.880800
2	0.872700	1.065535	54.449700	30.310200	49.116600	49.114600
3	0.533500	1.127618	56.591500	32.402100	50.907100	50.888200
4	0.281400	1.320929	55.819400	32.157200	50.336300	50.309800
5	0.123600	1.602202	55.500300	31.515700	49.945100	49.925600

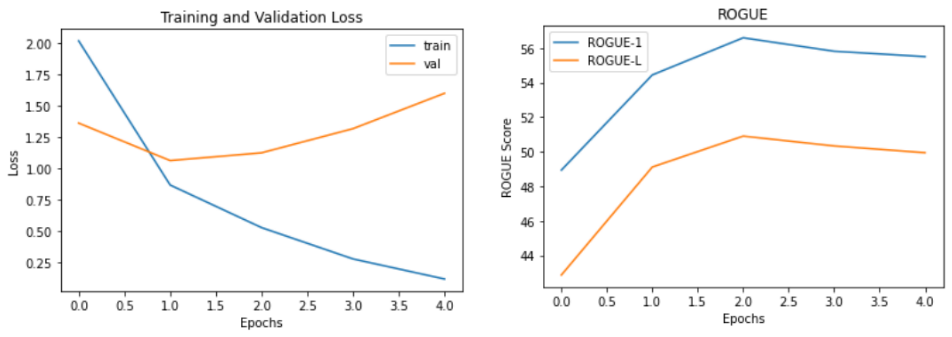


Figure 4.9: SubTask 2 Evaluation

4.5. Results and Analysis

We have tried to evaluate our methodology against the one described in the paper under the same environment. We have also evaluated our approach against a new methodology proposed by Google Research, Brain Team in the paper *Self-Consistency Improves Chain of Thought Reasoning in Language Models* which has been released on 6th April 2022 [8]**.**

The accuracies achieved by us are not the same as achieved in the papers, due to the difference in the scale of the models used by us and the one mentioned in the papers. But as describe in the paper "*Unsurprisingly, we see that the 175B model significantly outperforms the smaller models. Assuming a log-linear trend, we can naively extrapolate these results to estimate that a model with 10^{16} parameters would be required to reach an 80% solve rate, when using the full GSM8K training set*".

Table 4.1: Comparing Different Methodologies

Methodology	**Number of Parameters**	**Accuracy**
Our Method (mBART)	535M	8.5%
OpenAI (GPT-2)	350M	3.9%
Google (GPT-2)	350M	2.8%
Open AI (GPT-3)	6B	20%
Open AI (GPT-3)	175B	55%
Google (LaMDA)	137B	27.7%
Google (PaLM)	540B	74.4%

From Table 4.1 we see that accuracy increases with increase in the number of parameters. Due to the limits of computation and memory, we could only train our model with 535 million parameters but achieved a considerable boost in accuracy as compared to OpenAI GPT-2 model with 350 million parameters.

Google’s PaLM and LaMDA are the latest models trained on GSM8K dataset with 540 billion and 137 billion parameters respectively [8]. They have achieved 27.7% and 74.4% accuracies respectively. They have used different approach in improving the multi-reasoning ability of language models as discussed in Chapter 5.

So as described above assuming a log-linear trend between the accuracy and number of parameters in the model we can extrapolate that our methodology will perform at par with the methods above.

CHAPTER 5
CONCLUSION AND FUTURE SCOPE

5.1. Conclusion

We have seen that instead of asking the model to solve the entire question at once, breaking down the problem into sub-questions and then solving them provides a significant performance boost relative to a fine-tuning baseline.

We have trained our model within the scope of memory size and computation available. We have compared our results with models trained with more number of parameters on greater memory and computation architecture. We have shown that our proposed method will perform at par with other methods if we increase the number of parameters and computation memory.

5.2. Future Scope

Research conducted in this domain of solving Maths word problems has seen a rising trend. Google's PaLM trained on 540 billion parameters gives an accuracy of 74.4% while LaMDA gives an accuracy of 27.7% when trained with 137 billion parameters [8].

Recent methods use chain of thought prompting to accurately solve Maths Word Problems involving multi-step reasoning. The idea behind such an approach is similar to our methodology to break down problems into sub-problems and solve them to get a more accurate answer [8][9].

5.3. Programming Tools and Environments Used

- Exploratory Data Analysis – Python-Pandas, Numpy, Matplotlib
- Model Training and Evaluation – Dataset, Transformers, PyTorch
- IDE - Google Colab, Jupyter Notebook

REFERENCES

1. Cobbe, K., Kosaraju, V., Bavarian, M., Hilton, J., Nakano, R., Hesse, C., & Schulman, J. (2021). Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.
2. OpenAI Blog- Solving Math Word Problems (openai.com)
3. Dataset - github.com/openai/grade-school-math/grade_school_math/data
4. Devlin, Jacob, et al. "Bert: Pre-training of deep bidirectional transformers for language understanding." *arXiv preprint arXiv:1810.04805* (2018).
5. Yibin, Yao, et al. "Accuracy assessment and analysis for GPT2." *Acta Geodaetica et Cartographica Sinica* 44.7 (2015): 726.
6. A. Amini et. al. Mathqa: Towards interpretable math word problem solving with operation based formalisms. arXiv preprint arXiv:1905.13319, 2019.
7. T. B. Brown et al. Language models are few-shot learners. arXiv preprint arXiv:2005.14165, 2020.
8. Wei, J., Wang, X., Schuurmans, D., Bosma, M., Chi, E., Le, Q., & Zhou, D. (2022). Chain of thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903*.
9. Kojima, T., Gu, S. S., Reid, M., Matsuo, Y., & Iwasawa, Y. (2022). Large Language Models are Zero-Shot Reasoners. *arXiv preprint arXiv:2205.11916*.
10. Transformers BART Model Explained for Text Summarization (projectpro.io)
11. MBart and MBart-50 (huggingface.co)
12. A Visual Guide to Using BERT for the First Time – Jay Alammar – Visualizing machine learning one concept at a time. (jalammar.github.io)
13. The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time. (jalammar.github.io)
14. Chorowski, Jan K., et al. "Attention-based models for speech recognition." *Advances in neural information processing systems* 28 (2015)
15. BERT - https://towardsdatascience.com/keeping-up-with-the-berts-5b7beb92766
16. https://www.analyticsvidhya.com/blog/2021/06/why-and-how-to-use-bert-for-nlp-text-classification/
17. GPT-2 HuggingFace - gpt2 · Hugging Face
18. The Illustrated GPT-2 (Visualizing Transformer Language Models) – Jay Alammar – Visualizing machine learning one concept at a time. (jalammar.github.io)
19. Chain of Thought in Multi-Step Reasoning - 2201.11903.pdf (arxiv.org)
20. ROUGE Score - Measure NLP Accuracy With ROUGE | Towards Data Science
21. Deep Learning Stanford University - CS230: Deep Learning
22. [1910.10683] Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer (arxiv.org)
23. language_understanding_paper.pdf (openai.com)
24. [1810.04805] BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding (arxiv.org)
25. Brief Introduction to Attention Models | by Abhishek Singh | Towards Data Science
26. Transformers In NLP | State-Of-The-Art-Models (analyticsvidhya.com)
27. Transfer Learning - www.datacamp.com/community/tutorials/transfer-learning
28. Chorowski, Jan K., et al. "Attention-based models for speech recognition." *Advances in neural information processing systems* 28 (2015).