Machine Learning applied in the context of Question Generation and Question Answering

MEng (Hons) Final Year Project Report
May 2021

Chloe Thompson

Queen's University Belfast, Belfast, United Kingdom
cthompson68@qub.ac.uk

I declare that this report is my original work except where stated.

.....*CThompson*.....

Abstract

Question Answering and Question Generation is a prominent research problem and focus area in the space of Natural Language Processing (NLP). There have been a number of developments in models in the past years, with the introduction of the approach using Embeddings and Transformers. This paper aims to understand and demonstrate the use of NLP models, specifically BERT and GPT-2, in Question Answering and Question Generation tasks respectively, and how the training data used affects the outcomes. The results focus on the of F1 and Exact Match scores for the Question Answering models and BLEU and Meteor scores for both Question Generation and Question Answering models. Both model types are trained on two datasets for comparison. We perform hyper parameter tuning across BERT for Question Answering and GPT-2 for Question generation, with both datasets of Stanford Question Answering Dataset (SQuAD 2.0) and Question Answering in Context (QuAC). Further to the metrics outlined, we analyse the language these models perform strongly on, that being the types of questions that are able to be answered. Our best results for Question Answering were a BERT base model with a softmax output layer achieving an F1 score of 71.84 on the evaluation set. In Question Generation our best outcome for the GPT-2 model trained on custom tokens, resulted in only 10.88% of questions left unanswered. We also perform visualisations, analysis of clustering and donut plots over the question types to show the areas of improvement for model understanding.

1 Project Specification

Question Generation (QG) and Question Answering (QA) are NLP tasks focusing on the ability to automatically generate questions, and answers to questions, posed in natural language, respectively. Leveraging NLP techniques, it is aimed to develop a Question Generation and Question-Answering system that should be able to consume (un)structured text and, by capturing relevant information, return a set of comprehensive questions and answers.

Objectives

1. Familiarization with NLP techniques such as Named-entity Recognition (NER) and Relation Extraction (RE) leveraging Python & Java packages/libraries.
2. Identify meaningful data for QG and QA training sets.
3. Data Cleaning and pre-processing (tokenization, stop words, lemmatisation, stemming).
4. Develop a Question Answering model leveraging the techniques currently available with open source tools.
5. Evaluate the accuracy of the developed QG & QA models and benchmark the effect on performance of varying pre-processing techniques.

MEng Extension

1. Using machine learning techniques, being able to detect patterns within natural language and sentence structure.
2. Create a software tool in Python to perform document ingestion, pre-processing and produce QG/QA results.

Learning Outcomes

Upon completion of the project you will expect to have:

1. A comprehensive understanding of different machine learning methods (supervised and unsupervised).
2. A thorough understanding of NLP techniques, particularly QG & QA models
3. A substantial knowledge of Python programming.
4. Ability to handle large datasets and perform necessary processing of the data.
5. A working knowledge of development pipeline tools within AWS platform.

2 Introduction

The process of textual generation has been a focus within Natural Language Processing (NLP) for a number of years, which is the basis for tasks such as Question Answering and Question Generation. This project is not only designed to cultivate an understanding of NLP within Question Answering (QA) and Question Generation (QG), but to also evaluate and develop these practices by applying machine learning within a cloud-based environment.

For the development of the QA model, we focus on Google's BERT architecture (Devlin et al., 2019), as it is one of the leading architectures for language modelling (Devlin and Chang, 2018) and has shown great success as the backbone for a variety of NLP models and tasks. Looking at the QG model, GPT-2 created by AllenNLP (Radford et al., 2019) will be focused on, as while its architecture is not novel, being similar to that of a decoder only Transformer, it is leading within NLP due to both the model size and dataset size it was trained on. These two model implementations will allow further exploration of the underlying NLP theory within the Transformer architecture, enabling improvements on the base models if deemed necessary.

Two datasets are considered for initial training and analysis, Stanford Question and Answering Dataset (SQuAD 2.0) and Question and Answering in Context (QuAC). We will train QA and QG models with the dataset context paragraphs and ground truth answers or ground truth questions, respectively. During evaluation stages the metrics of Exact Match, F1, Bleu and Meteor score will be considered to compare our models under two different datasets and the underlying effect on model performance. Leveraging these initial results, further model developments via hyperparameter tuning can be explored in order to enhance model performance and lead to better evaluation results.

In order to understand the types of questions being generated and answered by the models we consider what is known as the special or W-questions. During this analysis we evaluate the models ability to answer Who, What, Where, When and Why questions, as well as Which and How based questions; any that are not this type are considered undetermined. In performing this subset analysis, we gain an understanding of the question styles the models are strongest in answering. This is done through graphing the question types and visualising the breakdown of question types.

In section 3, we start by outlining the background knowledge and methods within NLP that have led to the development of language modelling techniques and sophisticated models. This is followed by an analysis of prominent language models, namely BERT and GPT-2, along with their Transformer architectures. The design of datasets that are prevalent within the space of Question Answering and Question Generation, including a breakdown of dataset content and example snippets, will also be explored as part of the Literature Review. Section 4 details the methodology followed in this research, exploring the development and integration of QG and QA. The evaluation and subsequent results are detailed in section 5, followed by conclusions on the current state of work and outline future work in section 6. The project Gantt Chart can be seen in Section 8.1

3 Literature Review

3.1 Language Modelling

Language modelling is the use of mathematical processes to determine the probability of a given sequence of words, occurring within a sentence. These models are predominantly used for Natural Language Processing (NLP), where generated or predicted text is given a probability which denotes the likelihood of that sentence occurring. The statistical model is trained on a large corpus and establishes an understanding of the order that words are most likely to occur. This model can then be used for word predictions or generation of new sentences, based on the learned features. Due to language modelling implementing probabilistic word predictions, it is used mostly for tasks such as Machine Translation or Question Answering. The mathematical formulas for statistical language models enable a variation in results, depending on the number of words considered for context when determining a probability. Techniques such as Lemmatization and Stemming (Manning et al., 2008) are often considered preceding the

use of Language models, to train models on the root forms of words. While language modelling has a number of uses, such as Part-of-Speech Tagging (POS) (Manning and Schutze, 1999) and Named-Entity Recognition (NER) (Bird et al., 2009), the in-depth explanation of these statistical approaches and implementation are out of the scope for this project and will not be covered.

3.2 Natural Language Processing Models

To understand the concept of models for Natural Language Processing, it is first important to understand the expected inputs and outputs of these models. The inputs for NLP models are created based on the fundamental concepts of Language Modelling. These techniques take a sentence of words and transform it into a sequence of vectors. This transformation is called encoding and involves using word embeddings to match a word to a discrete space, where words of similar semantic meaning have close vector representations. Following the model training, the resulting vectors are decoded, the inverse operation of encoding, to produce new sentences or labels for the input; note that some encoding or decoding stages are not always required which is the case for BERT and GPT-2 implementations.

Neural Networks (NN) were initially used for NLP, where input vectors are passed through network layers, which perform mathematical transformations with weights applied and output new representations from the model. NN's are Feed Forward networks (Dongare et al., 2012), trained on continuous representations, known as embeddings, which differentiate them from statistical language models; however, these inputs are required as a fixed length. By considering backpropagation with Recurrent Neural Networks (RNN) (Dongare et al., 2012), the issues of fixed length inputs could be solved. The approach with RNN's allowed for inputs of varying length and reduced the number of parameters, as the propagation shared weights among the layers. In implementing back propagation the issue of vanishing gradients appeared, where an RNN struggles with learning long-term dependencies for an input sequence. With the shortcomings of NN's and RNN's, the Long Short-Term Memory (LSTM) model was developed (Hochreiter and Schmidhuber, 1997), which solved the issue of vanishing gradients for RNN's. Although the LSTM performs well, the training time is extensive due to it considering all the words in the vocabulary before updating layer weights and employs the feed forward method therefore losing context. The approaches taken over the years have been summarised by Jing and Xu (2019). The loss of context with LSTM was tackled by ELMo through the use of bi-directional LSTM.

Embeddings from Language Models (ELMo) was developed by Peters et al. (2018) within AllenNLP, and is a novel way to represent word vectors or embeddings. This development created a bi-directional language model, using an LSTM, that enabled the context of the word within a sentence to affect the resulting embedding; this is important due to polysemy, where words can have multiple meanings. Figure 1 shows a high level view of the ELMo approach, where a sentence is input into the model and the outputs are the embeddings for each of the words within the inputted sentence. Inside the ELMo model is the bi-directional LSTM; this is built from a forward language model and a backward language model. These two inner models produce their vector embeddings for each internal state which are concatenated together, then multiplied by the model weights and finally summed for the output embedding. The final embedding is multiplied by a gamma value which aids the result, due to only using the last layer of the model for context. The development of ELMo produced outstanding results in comparison to that of neural models at the time, see Figure 2, where a direct improvement could be seen in the F1 results by incorporating ELMo in the neural models. This was the foundation for the development of sophisticated language models such as Transformer based models.

Google continued the internal joke naming convention of Sesame Street models with the development of BERT (Bi-directional Encoder Representations from Transformers) (Devlin et al., 2019), which built on the previous work of ELMo in contextual representations. BERT was developed to tackle the limitations of previous models, such as including the past and future contexts of the target word in the language model. While ELMo, like BERT, implemented bi-directionality it allowed the word

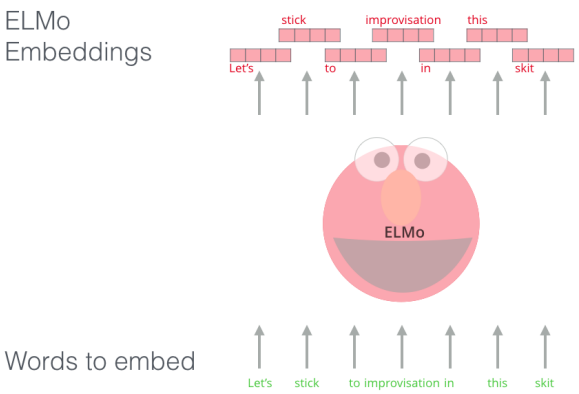

Figure 1: ELMo Embedding Architecture from (Alammar, 2018)

Task	Previous SOTA		Our baseline	ELMo + baseline	Increase (absolute/ relative)
SQuAD	Liu et al. (2017)	84.4	81.1	85.8	4.7 / 24.9%
SNLI	Chen et al. (2017)	88.6	88.0	88.7 ± 0.17	0.7 / 5.8%
SRL	He et al. (2017)	81.7	81.4	84.6	3.2 / 17.2%
Coref	Lee et al. (2017)	67.2	67.2	70.4	3.2 / 9.8%
NER	Peters et al. (2017)	91.93 ± 0.19	90.15	92.22 ± 0.10	2.06 / 21%
SST-5	McCann et al. (2017)	53.7	51.4	54.7 ± 0.5	3.3 / 6.8%

Figure 2: ELMo enhanced neural models Performance Comparison Table from (Peters et al., 2018)

being predicted to 'see itself' in a multi-layer model. The innovation with BERT was in the replacing of language modelling with the concept of Masked Language Modelling; a new technique that used probability to mask a certain number of words within the sentence.

The development of Transformers was the basis for creating the architecture for both BERT and GPT-2 models. Transformer architecture was developed by Google (Vaswani et al., 2017) and contains a number of encoder layers followed by a number of decoder layers, being six of each in the paper, shown in Figure 4. A single encoder within the Transformer contains a self-attention layer and feed forward neural network, see Figure 3. A single decoder within the Transformer contains a self-attention layer, followed by encoder-decoder attention and a feed forward neural network, see Figure 3. The encoder-decoder attention is calculated using a formatted output from the encoder layers into two attention vectors, these are used to help the decoder focus on certain areas of the sentence. The self-attention and feed forward neural network are identical in both the encoder and decoder. The self-attention layer takes the input sequence and encodes it, word by word, where each word embedding depends on the strength of relations with those surrounding it. These are passed to a feed forward neural network which generates a new non linear representation. A final linear and softmax layer take the output vector and produce a final word vector. The linear layer is a fully connected neural network that projects the vector into a larger space, creating a new vector referred to as a logits vector. This logits vector is converted into a number of word probabilities by the softmax layer, with the highest probability being chosen as the output word.

A Transformer takes input sentences, which have been formatted into embeddings. These input embeddings are created by taking the input sentence, tokenizing it with a word piece tokenizer and adding position embeddings, which encode word order. When encoding sentences language models can make use of special tokens that denote a number of properties in the sentence. These tokens can show the start of the sentence, end of the sentence or the boundary between sentences (i.e. a separation token showing where sentence A ends and sentence B begins). Tokens are included before the encoding of sentences occurs and vary between models. The Transformers for BERT add segment embeddings through tokens during encoding which distinguish the sentence or between the two sentences and GPT-2 uses set or custom tokens to indicate the separation within encoded text.

The Transformer architecture for BERT only contains encoder layers of the outlined basis for a Transformer. These encoders like the standard Transformer contain self-attention layers and feed forward neural networks. The number of layers within the BERT Transformer creates the differentiation in BERT Base and BERT Large models; specifically BERT Base contains 12 and BERT Large contains 24 encoder layers within their Transformers. BERT uses specific tokens to indicate the start, end and separation between sentences. The tokens are a requirement and are for both single and two sentence inputs; however the separation token is not always needed if a single sentence is being passed. The [CLS] token stands for classification and denotes the start of the sentence, whereas the [SEP] token marks the end of the sentence for a single sentence, or the separation of two sentences. BERT makes use of

a special [MASK] token that is used to hide the word in a sentence that is being predicted by the model.

BERT models are pre-trained in two ways simultaneously to learn representations of language. These are Masked Language Modelling (MLM) and Next Sentence Prediction (NSP). For MLM the model is fed a sentence with mask tokens replacing a set percentage, 15%, of words and the model predicts these hidden words. For NSP, the model takes two sentences and predicts whether the second sentence follows the first, enabling the model to learn about inter-sentence dependencies and context within language.

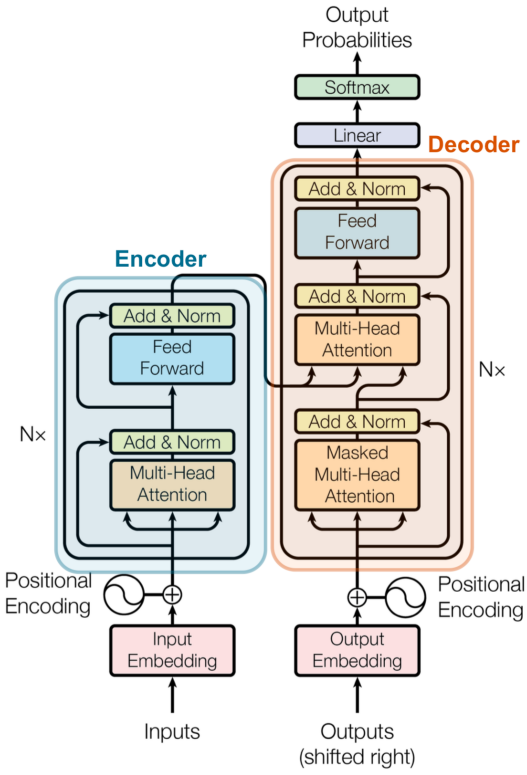

Figure 3: Transformer Architecture from (Vaswani et al., 2017)

Once vectors have been obtained from BERT a final output representation is produced. Each word vector at the output is passed to a feed-forward neural network (FFNN) simultaneously, which contains a softmax function that converts the FFNN output of a logit vector into probabilities. Out of the probabilities, the cell with the highest probability is then chosen as the final word prediction or binary output for Next Sentence Prediction. A full example is displayed in Figure

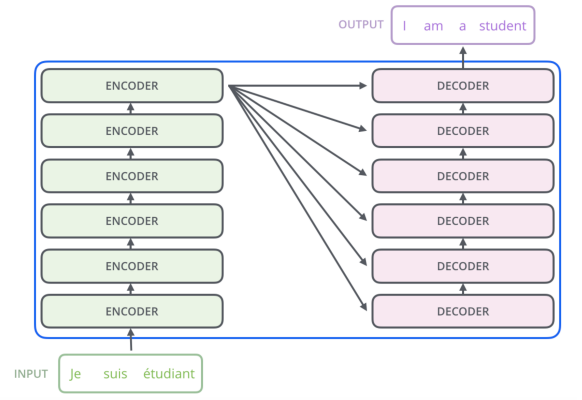

Figure 4: High Level Transformer from (Alammar, 2019)

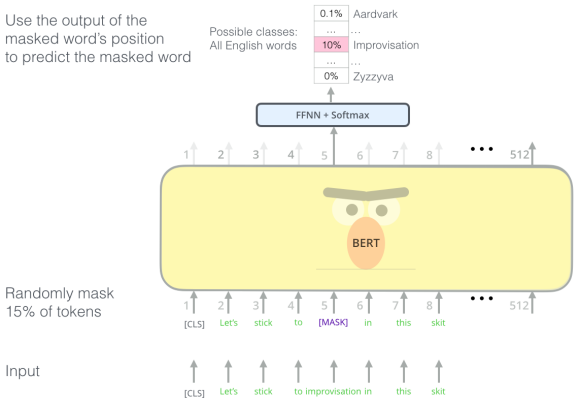

Figure 5: BERT Illustrated Diagram from (Alammar, 2018)

5, from the illustrated explanation by Alammar (2018).

With the popularity of BERT, Google released different versions of BERT, BERT Base and BERT Large, using different numbers of transformer layers (and therefore parameters). These versions, namely RoBERTa (Liu et al., 2019), DistilBERT (Sanh et al., 2020) and ALBERT (Lan et al., 2020), contain subtle differences to their predecessor BERT. These models were pre-trained and contain a large number of encoder layers, making up their Transformer architecture. In this work, we will focus on BERT for QA, as this is the most well-established of the family and can be appropriately trained for the needs of this research.

The Transformer architecture for GPT-2 only contains decoder layers from the original Transformer proposed; however it also makes a change to the inner workings of a decoder. The decoder within

the GPT-2 Transformer is created with two layers of Masked Self Attention and a feed-forward neural network, removing the encoder-decoder attention layer. The masked self-attention in the GPT-2 layer is different to that of BERT in the fact that it doesn't make use of masked tokens. Masked self-attention is a variation on self-attention, see Figure 6, where the attention block is prevented from looking at tokens following the current token being predicted by weightings. GPT-2 comes in four size variations, small, medium, large and extra large with 12, 24, 36 and 48 decoder blocks in a Transformer, respectively.

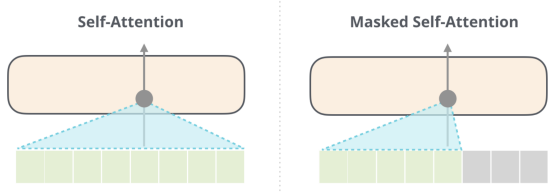

Figure 6: Transformer Architecture from (Alammar, 2019)

In order to train GPT-2 the input data needs to be formatted to contain GPT-2 specific tokens, there is the possibility to add additional custom tokens depending if your training text has specific patterns. These tokens help the model identify where to generate text from. The base token that is recognised in pre-trained GPT-2 is, $< |endoftext| >$, which denotes the start of the text section. When generating text the token is passed as an input and the model will generate up to a certain length following the token. GPT-2 generates one token at a time in an auto-regressive manner, meaning the next token being generated considers all previous tokens inclusive of the most recent generated token.

When generating text, the starting token $< |endoftext| >$ or custom token is passed to GPT-2 as the initial token, this is then passed through the decoder layers, with the output resulting in a vector. Each row entry in this output vector corresponds to a word embedding known in the models vocabulary; the token with the highest probability is then chosen as the final word output. The model at this point has completed an iteration by generating the next word and this token is then used to aid the generation of the next. This process continues until either the length of output has been generated or an end token is generated. An example of this generation can be seen in Figure 7, where the $ symbol denotes where the GPT-2 model is generating text from.

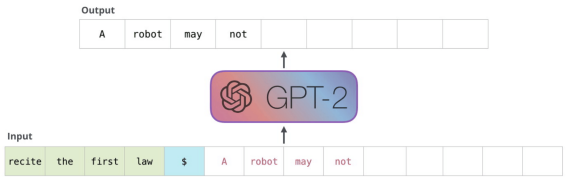

Figure 7: Transformer Architecture from (Alammar, 2019)

With the success of GPT-2 models in the generational text space, OpenAI continued improving the models with the subsequent development of GPT-3. GPT-3 has 175 Billion parameters, in comparison to the 1.5 Billion in the largest GPT-2 model. This latest model is vast and still under Beta release, which is why the focus will be on that of GPT-2.

There are a number of prominent NLP tasks and research problems to be solved using both of these models, BERT and GPT-2; that being Question Answering and Question Generation. QA & QG revolve around having both a large corpus of knowledge to pull from and the models' ability to represent the grammatical structure and lexical semantics of a sentence. Having both of these in place is the grounding for a model to produce either a grammatically sensible answer or question.

3.3 Datasets

As the area of NLP became more prominent, a number of research institutions created datasets for specific problems, using crowdsourcing to generate labelled data. Question Answering and Question Generation has proved an interesting challenge for those in the NLP space and because of this, there have been a number of datasets created to compare latest developments. Most notably the SQuAD 2.0 dataset from Stanford (Rajpurkar et al., 2016) has been a staple within Question Answering and Question Generation. The Stanford Question Answering Dataset (SQuAD) was built through crowdsourcing answers to over 100,000 questions on Wikipedia articles, where the answers are a span of text within the article. Following from the development of SQuAD, a number of

Dataset	Multi turn	Text-based	Dialog Acts	Simple Evaluation	Unanswerable Questions	Asker Can't See Evidence
QuAC	✔	✔	✔	✔	✔	✔
CoQA (Reddy et al., 2018)	✔	✔	✗	✔	✔	✗
CSQA (Saha et al., 2018)	✔	✗	✗	✗	✔	✗
CQA (Talmor and Berant, 2018)	✔	✔	✗	✔	✗	✔
SQA (Iyyer et al., 2017)	✔	✗	✗	✔	✗	✗
NarrativeQA (Kociský et al., 2017)	✗	✔	✗	✗	✗	✔
TriviaQA (Joshi et al., 2017)	✗	✔	✗	✔	✗	✔
SQuAD 2.0 (Rajpurkar et al., 2018)	✗	✔	✗	✔	✔	✗
MS Marco (Nguyen et al., 2016)	✗	✔	✗	✗	✔	✔
NewsQA (Trischler et al., 2016)	✗	✔	✗	✔	✔	✔

Figure 8: Dataset Feature comparison from (Choi et al., 2018)

datasets were produced such as QuAC (Choi et al., 2018), using the same crowdsourced responses to questions approach. These were in a similar structure to SQuAD and contains three focus areas: unanswerable questions, multi-turn interactions and abstractive answers. Figure 8, shows the comparison of the QuAC dataset from Choi et al. (2018) in relation to other popular datasets for NLP, inclusive of the newer version of SQuAD.

Analysing this table shows that in a number of areas, QuAC contains question types that SQuAD does not, in particular multi-turn, dialog acts and asker can't seek evidence questions. All of these features present in QuAC are related to dialog and multi question and answer situations, where an answer needs to be questioned to further understanding. Figures 9 & 10 show examples of the question and answer data contained in SQuAD and QuAC. In order to understand the performance difference across these datasets, a qualitative study was conducted (Yatskar, 2018). Due to the QuAC and CoQA datasets containing similar abstractive answers, models that were trained on this data were seen to be easily adapted over the two datasets. Models that were trained on a dataset tended to not act performantly on testing data from other sets, which is to be expected due to the content of knowledge paragraphs differing over the datasets. However, it can be noted that pre-training and re-training with two separate datasets did enable better performance on predicting with the model on the second dataset. As the CoQA and QuAC datasets were similar in their abstractive questions, only QuAC was brought forward in the later stages of research to be compared with models trained on SQuAD.

In meteorology, precipitation is any product of the condensation of atmospheric water vapor that falls under **gravity**. The main forms of precipitation include drizzle, rain, sleet, snow, **graupel** and hail... Precipitation forms as smaller droplets coalesce via collision with other rain drops or ice crystals **within a cloud**. Short, intense periods of rain in scattered locations are called "showers".

What causes precipitation to fall?
gravity

What is another main form of precipitation besides drizzle, rain, snow, sleet and hail?
graupel

Where do water droplets collide with ice crystals to form precipitation?
within a cloud

Figure 9: Data Example from SQuAD paper (Rajpurkar et al., 2016)

3.4 Question Generation

Question Generation is the task of generating questions given a context paragraph and in some cases, a specified answer within the paragraph. There have been a number of papers written about the area of Question Generation (QG) over the past few years. An initial study from Zhou et al. (2017), provided an extensive comparison on techniques which would improve the quality of sensible question generation; through combining a range of preprocessing and token insertion methods as inputs for a Neural Question Framework (NQG) (i.e. a model composed of encoders and decoders). The input processing considered to create the word vectors included, lexical features,

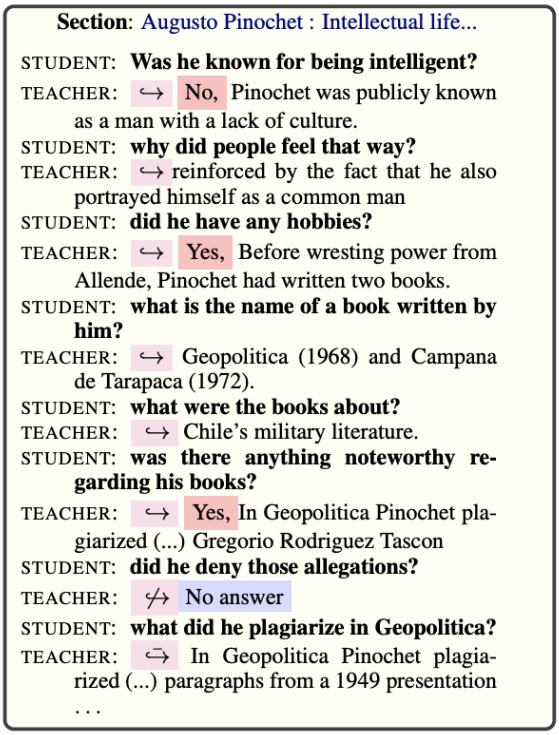

Figure 10: Data Example from QuAC paper (Choi et al., 2018)

embeddings, NER, POS tagging and the answer position feature which notes the span of the answer location within sentences. A copy mechanism was discussed, which denoted the probability of copying words from the input sentences into the answers; this approach is designed to help with rare or unknown words. In summary, their models had a strong performance on who, what, when and how questions but performed less optimally on questions relating to which. This NQG Framework was compared to that of Du et al. (2017), on performance considering sentence based and paragraph based models, while introducing the technique of over-generate-and-rank. This technique allows for multiple questions to be generated from an input sequence, which are ranked to allow for trimming down to an acceptable set of questions. The sentence level model used the encoder sentence representation to initialise the hidden state for the decoder in the LSTM. Whereas the paragraph level became a concatenation of the sentence encoder output and paragraph encoder output, that contained a length threshold for truncating at a certain length. With this analysis it was concluded by Du et al. (2017), the paragraph model was not fully performant across all question categories but achieved strong results for questions outside of the sentence, proposing the introduction of a copy mechanism for dialogue as a method to improve the questions generation process.

With the progression in models and the release of BERT between 2017 and 2019, question generation took a leap forward. The ability to fine-tune a pre-trained model for specific NLP tasks lead to many studies for Question Generation, including that of Chan and Fan (2019). This paper references the strong work of NQG for Reading Comprehension, in relation to their paragraph model finding that paragraph context can improve QG performance. In their work the authors adapted the BERT base model considering two different approaches, a BERT model trained on input SQuAD data and another that used Sequence Question Generation (SQG), where the previous decoded results are taken into consideration for decoding a current embedding. The findings noted that the standard BERT-QG model performed poorly in comparison to those preceding BERT development; however, the inclusion of the previous decoded tokens in BERT-SQG produced state-of-the-art results. Within their work the authors showed that the inclusion of paragraph-level contexts can improve performance and provided a strong baseline for further research.

As the BERT family developed, Lewis et al. (2019) introduced the denoising autoencoder for pretraining sequence-to-sequence models (BART), as a development on BERT and RoBERTa. BART (Lewis et al., 2019), as a denoising auto-encoder model, is trained on corrupted text that has been created using noise techniques such as token deletion, token masking and sentence shuffling; with this approach the aim is to reconstruct the original text. The flexibility on the type of noise introduced to sentences is where the strong performance of BART is seen, with the best approach found to be the random shuffling of sentences within a paragraph combined with one of the forms of token noise, be that deletion or masking. The approach of sentence shuffling encourages the model to generalise for NSP and consider a wider span of sentences for context. Most interestingly BART is an adaptation of not only BERT but GPT-2, with the encoder being that

from BERT and the decoder taken from GPT-2. The authors concluded that the BART models' performance on specific tasks, when pre-trained, could be greater than that of BERT, especially on that of Question Generation.

A similar approach to that of BART (Lewis et al., 2019) was taken by (Klein and Nabi, 2019) in which a QA to QG pipeline was developed through connecting BERT and GPT-2 for the tasks respectively. The authors investigated tying together BERT and GPT-2 in an end-to-end fashion for training and the impact that a QA model could have on the quality of QG quality. In their work the QG model was adapted based on the QA effectiveness, where the loss from the answering was back-propagated through to the QG model, in order to improve the generation and outcome for unanswered questions. The correct questions are repeatedly prompted as a replay mechanism, to prevent forgetting. Their findings showed that the questions generated via the feedback from the QA model were of a higher quality than that of the standard GPT-2 language model; determining that the feedback loop attributed to a stronger understanding of complex relationships in sentences by the QA model. Future developments outlined that the improvement of QG model and reduction of annotated data being required would be viable next steps, given the achievement of a BERT and GPT-2 pipeline.

3.5 Question Answering

Similar to Question Generation (QG), the task of Question Answering (QA) has been a developing area of NLP. Both problems can be seen as interrelated, with the approach for either being beneficial for implementing the other. This relationship was explored by Wang et al. (2017), on how QG and QA approaches could be interlinked. In order to compare the performance of a joint model, two baseline models for generation and answering were implemented, based on a sequence-to-sequence model with Bi-directional LSTM encoder and RNN decoder. To create the interlinked Question Answering model, the same sequence-to-sequence model approach is taken, using joint training to feed data into the model. This joint training method alternates between feeding the model Question Answering and Question Generation data. The authors concluded that their joint model outperformed the QA only model, but only by a small margin, however, the joint model results were lower than that of state-of-the-art QA models. Despite this the results highlighted the effectiveness of joint training and concluded that it would be an area of future development for Question Answering.

As we have already seen in the case of Question Generation (Klein and Nabi, 2019), the introduction of the pre-trained BERT model has proven useful for the specific task of Question Answering. Zhang and Xu (2019) introduced the challenge of natural language comprehension tasks and the approach of fine-tuning layers within the BERT base model to tune for Question Answering. The study compares the adapted model to the standard BERT model, trained for QA and outlines the number of encoder and decoder variations and output layers placed around BERT for their improvements. The authors concluded that adding encoder-decoder layers on top of BERT, including RNN variations of these, help mitigate against temporal dependencies. However, the implementation of additional self-attention and alternative output layers did not increase performance. This outlines that BERT as a base model is capable of having enhancements and further development for specific tasks in order to achieve performant results, although this may compromise the generalist nature of the model.

With the development of datasets in the QA space (Schwager and Solitario, 2019) explored the impact that the latest SQuAD dataset at the time, SQuAD 2.0, and tuned hyper parameters had on BERT model outcomes. The authors achieved competitive results through the adjustments in Learning Rate, Epochs and Dropout. Interestingly the work explored modifications to the loss function and whether varying the weighting to span made a difference to outcomes. They showed that weighting of the start of the span over the end of the span had little impact however, this led to a later development of weighting answer outcomes. By experimenting with the weights towards questions that had an answer it was realised that this had an increase for outcomes; the reverse of weighting questions without answers decreased performance. The authors concluded that an ensemble approach between models with a higher

weighting to the start span, weighting questions with answers more than those without answers and vice versa, produced competitive results with possible future work being the exploration of this with BERT large.

It is clear from the literature that there is a benefit in using pre-trained language models. The work of (Alberti et al., 2019), explored the use of a pre-trained BERT SQuAD model on the Natural Questions (NQ) dataset (Kwiatkowski et al., 2019). The task differs to that of SQuAD due to the model being required to read the full article text to find the 'long answer', the paragraph containing the answer, and following this find the 'short answer', the actual answer to the question. The authors introduced special tokens to indicate the table, list or paragraph number that the model is looking at in the longer article, as the long text was split to fit the 512 token length input required for BERT. The outcome was a 30% improvement on long answers and 50% improvement on short answers for the NQ dataset and provided a basis for further developments on the dataset task. The work demonstrates the ability to transfer pre-trained BERT models for improvements on other or related tasks.

Within the papers mentioned comparisons are made to previous works, their models and the performance of these. A key evaluation metric is BLEU (Bilingual Evaluation Understudy), an algorithm in which the quality of machine translated text can be evaluated (Papineni et al., 2002). BLEU uses the n-grams (Jurafsky and Martin, 2019), summed and divided for an average, of the true sentence and compares them to the n-grams, summed and divided, for the generated text. However, the count is modified to reduce the chance of over generation of reasonable words, with a method introduced of unigram modified precision. In using BLEU, the model translation can be compared to that of multiple reference sentences. By incorporating this measure during evaluation of models, a common comparison can be made between both current work and human judgement.

As with all developments in models, the measurement of model performance with BLEU was improved upon with the introduction of Meteor (Banerjee and Lavie, 2005). Meteor was developed to address weaknesses outlined in the BLEU measure, such as the lack of recall or geometric averaging over n-grams that could result in zero if a single component was zero. The metric calculates the alignment of the translated sentence to the reference sentence through a number of stages to then produce a harmonic mean through the precision and recall. This harmonic mean is used to calculate the overall score which includes a penalty to account for longer translation sequences; as translations can over produce in the hope for more matching words. When scored on a system level to match the use case of BLEU it was presented that Meteor was a closer measure to that of human judgement while also being able to be applied at a corpus and sentence level.

4 Methodology

4.1 Models

In order to implement the BERT model, we needed an understanding of its architecture and mathematical concepts for NLP techniques, which was conducted through online training courses, covering both theoretical knowledge and the practical implementation. With this basis of understanding through exploratory learning, our implementation of the first version of BERT for Question Answering models took place, leveraging a Tensorflow and Keras Layers technique following the approach developed by Jocqueviel et al. (2020).

In taking this approach we inserted the pre-trained BERT model as a layer in a custom model class and a further layer was implemented on top of the BERT outputs, that would enable the fine-tuning for the task of Question Answering. This additional layer at the output was a dense layer that followed the outputs of a sequence from the BERT model layer. In implementing a dense layer, the final model outputs are the start and end positions, indicating the span of text in which the answer is present within the context paragraph. This approach provided a good understanding of the model architecture required. However, due to the custom nature of this approach and the computational times required to train models, following this approach was unfeasible.

To overcome time constrain limitations, we decide

to pursue the implementation of BERT using the HuggingFace pytorch approach, leveraging the included BERT QA feature and the ability to have multiple GPUs for training purposes. This implementation was identical to the Tensorflow package previously described, considering the BERT model construct with the addition of a linear layer on the output to produce start and end span outputs. This provided a function based script that was callable with parameters, with modifications made to enable iterations over hyper parameters.

BERT models make use of AdamW (Adam optimizer with weight decay), an Adam optimizer variant. This optimizer was first outlined by Loshchilov and Hutter (2019) and it was chosen due to its ability to aid model generalisation in weight regularization. We implemented the AdamW optimizer with custom learning rate and dropout, which will be described in later sections, see 4.4.2. The optimiser coupled with the loss function within training, enables weights to be iteratively updated until a minimum loss is found. The objective function to be minimised was the sparse categorical crossentropy loss (Huggingface, 2021), where the probabilities of the results are returned as a single true class value and the total loss is calculated as the mean of the start and end positions. In the case of BERT, sparse categorical crossentropy was used to produce a single ground truth position for the start and end positions, based on the tokens probability from the model output.

After understanding and implementing the transformer architecture for BERT, the implementation of GPT-2 for QG was conducted by leveraging the GPT-2 simple library[1]. As mentioned in 3.2 the GPT-2 model size is dependent on the number of decoder layers within the Transformer. We implemented GPT-2 model with 12 decoder layers or 124 Million parameters, initially leveraging the trained model checkpoints from the library. This implementation enabled the use of functions within the library to finetune and generate text with the fine tuned model as well as the setting up of hyperparameters. We implemented a set of custom tokens that were used within the text file to denote separations between the start of context paragraphs, questions and the start and end of the text for that iteration. In including these tokens we are training the model to identify areas within the text that will aid the model, this is relevant for the later generation stages when a question token is passed and the model generates the text following the token.

GPT-2 has two options for its training optimiser; Adam and SGD (Stochastic Gradient Descent). Our implementation considered Adam, the default optimizer setting in GPT-2 simple. The Adam optimiser for GPT-2 follows the same considerations as previously described for AdamW but with non-weighted decay (Loshchilov and Hutter, 2019). An adaptive learning rate was considered in order to prevent the optimizer from becoming stuck in a local minimum of the objective function. As in the BERT model, GPT-2 uses the cross-entropy loss as the Adam optimiser objective function, which calculates the probability of the output token between 0 and 1. As GPT-2 outputs a vector of tokens that are possible and a probability for each token in the vector, with the token of the highest probability being chosen as the generated word. With GPT-2 being an auto-regressive model, only a single token is outputted at a time.

4.2 Datasets

For the specific task of Question Answering there are a number of prominent datasets within the NLP space, one of which is SQuAD developed by Stanford. We chose this dataset due to the nature of popularity as a leading dataset within NLP and as a point of comparison to previous work within the field, to benchmark the results of this project. Similar to this, the dataset of QuAC was chosen as a comparative dataset to that of SQuAD, due to the addition of context related questions and answers within the dataset. An overview of dataset breakdown into training and testing (dev) sets can be seen in Table 1. The larger total for SQuAD is due the dataset being in its second iteration with further data being inserted into the original set, for further details see Rajpurkar et al. (2018).

The data within both QuAC and SQuAD contains context paragraphs, questions, an answer for the question and the span for the answer positions. On

[1] https://github.com/minimaxir/gpt-2-simple/tree/afe38c43c8b40d1e81d0a57ccfa1a3138e7626a0

our investigation of the data within both QuAC and SQuAD, it was noted that the json formatting, while extremely similar, contained differences in structure which would need standardised to a single format in advance of training for consistency with the structure required for both BERT and GPT-2 models. The script for the BERT models takes the input json files in the SQuAD json format, requiring the QuAC json to be manipulated to contain the correct fields in the same structure and order. For GPT-2 models the input data is required to be in a text or csv file, that contains any special or end of text tokens. For this both SQuAD and QuAC were altered to contain the same token and data format and saved out to text files. This will be outlined in detail in Section 4.4 covering our implementation.

	SQuAD 2.0	**QuAC**
Training	130,217	101,978
Testing (Dev)	11,864	18,711
Total	142,081	120,689

Table 1: Dataset Breakdown

Further to the differences in dataset structure, the two datasets differ in their context paragraphs and question and answer style. Within SQuAD the context paragraphs have an average length of 758 characters, with a maximum of 4063 and minimum of 151, meaning that there are shorter spans of text for the answer to be found within. The questions asked within SQuAD can also be deemed as more concise in nature, with the questions being asked gaining short and factual responses for the answer. When comparing this to QuAC it is clear that the paragraphs are longer, on average a length of 2407 characters, with a maximum of 9791 and minimum of 1361, which gives more room for context dependent questions and answers rather than factual responses. A number of the questions being asked in QuAC can be seen to be looser in their intent, causing the corresponding answer to be unknown or contain longer explanations that require contextual reasoning to determine. Sample snippets from both SQuAD and QuAC can be seen in Figures 11 and 12.

4.3 Evaluation

A comprehensive form of evaluation was established in order to compare the performance of the implemented models. This considered the outputs of each model and their specifications. An evaluation of model performance for BERT and GPT-2 is performed following the training stage on an evaluation set, enabling one function call to be made to the scripts with parameters passed for both training and evalution stages.

For the QA implementation using BERT we considered F1, the average overlap between the prediction and the ground truth answer, and Exact Match (EM), where the match metric considers the predictions that match any one of the ground truth answers exactly, as performance metrics. These metrics are analysed against human performance for the dataset, 82.3% for Exact Match and 91.2% for F1 with SQuAD (Rajpurkar et al., 2018) and 74.6% for F1 with QuAC (Choi et al., 2018).

In addition to the metrics of F1 and EM, we utilize BLEU Papineni et al. (2002) and Meteor (Banerjee and Lavie, 2005), to evaluate the quality of machine translated text from both the QA and QG models. These two metrics correlate with human reasoning and are commonly used to evaluate the quality of the output of text generation models. The use of both metrics is justified by the shortcomings of BLEU on machine translation tasks, such as question generation (Banerjee and Lavie, 2005) For example, BLEU will penalise generated sentences of a non-exact match between the reference and translation occurs; an issue common in morphologically rich languages. This shortcoming is addressed by Meteor, which includes the stems and synonyms in the evaluation, enabling creative translations to be considered as a good performance despite an exact match not being achieved. This combination is essential to evaluate the performance of the GPT-2 QG implementation in generating free text questions.

To implement BLEU, the NLTK implementation [2] of BLEU was considered. This enabled the evaluation of text over a number of n-grams and a choice of smoothing technique to prevent inflated precision values. By taking the BLEU

[2] https://www.nltk.org/api/nltk.translate.html

Context	Questions	Answers
The foundations of the British Empire were laid when England and Scotland were separate kingdoms. In 1496 King Henry VII of England, following the successes of Spain and Portugal in overseas exploration, commissioned John Cabot to lead a voyage to discover a route to Asia via the North Atlantic. Cabot sailed in 1497, five years after the European discovery of America, and although he successfully made landfall on the coast of Newfoundland (mistakenly believing, like Christopher Columbus, that he had reached Asia), there was no attempt to found a colony. Cabot led another voyage to the Americas the following year but nothing was heard of his ships again.	Who commissioned John Cabot's voyage?	King Henry VII of England
The New York City Fire Department (FDNY), provides fire protection, technical rescue, primary response to biological, chemical, and radioactive hazards, and emergency medical services for the five boroughs of New York City. The New York City Fire Department is the largest municipal fire department in the United States and the second largest in the world after the Tokyo Fire Department. The FDNY employs approximately 11,080 uniformed firefighters and over 3,300 uniformed EMTs and paramedics. The FDNY's motto is New York's Bravest.	How many firefighters work for the New York City Fire Department?	11,080
For maximum life, capacitors usually need to be able to handle the maximum amount of reversal that a system will experience. An AC circuit will experience 100% voltage reversal, while under-damped DC circuits will experience less than 100%. Reversal creates excess electric fields in the dielectric, causes excess heating of both the dielectric and the conductors, and can dramatically shorten the life expectancy of the capacitor. Reversal ratings will often affect the design considerations for the capacitor, from the choice of dielectric materials and voltage ratings to the types of internal connections used.	When designing a capacitor, what is an important rating to consider?	Reversal ratings
The first blue-violet LED using magnesium-doped gallium nitride was made at Stanford University in 1972 by Herb Maruska and Wally Rhines, doctoral students in materials science and engineering. At the time Maruska was on leave from RCA Laboratories, where he collaborated with Jacques Pankove on related work. In 1971, the year after Maruska left for Stanford, his RCA colleagues Pankove and Ed Miller demonstrated the first blue electroluminescence from zinc-doped gallium nitride, though the subsequent device Pankove and Miller built, the first actual gallium nitride light-emitting diode, emitted green light. In 1974 the U.S. Patent Office awarded Maruska, Rhines and Stanford professor David Stevenson a patent for their work in 1972 (U.S. Patent US3819974 A) and today magnesium-doping of gallium nitride continues to be the basis for all commercial blue LEDs and laser diodes. These devices built in the early 1970s had too little light output to be of practical use and research into gallium nitride devices slowed. In August 1989, Cree introduced the first commercially available blue LED based on the indirect bandgap semiconductor, silicon carbide (SiC). SiC LEDs had very low efficiency, no more than about 0.03%, but did emit in the blue portion of the visible light spectrum.[citation needed]	What substance helped demonstrate the first blue electroluminescence?	zinc-doped gallium nitride
The first helicopter landing at the new airfield was conducted by the Wildcat HMA.2 ZZ377 from 825 Squadron 201 Flight, embarked on visiting HMS Lancaster on 23 October 2015.	Where did the helicopter embark on visiting?	HMS Lancaster

Figure 11: SQuAD Dataset Sample

scores of 1-4 n-gram lengths, we were able to determine the measure that was the best judgement for the text span calculated. We selected the smoothing method of 4, as this measure smooths the counts for shorter translations that may inflate the precision measure (Chen and Cherry, 2014). This is due to the answer spans being shorter for SQuAD datasets in comparison to QuAC.

The BLEU measure is calculated as a modified n-gram precision, shown in Equation 4.1, where p_n is the modified precision for the n-grams, w_n is the weights that sum to 1, 4.2 BP is the brevity penalty that penalises shorter translations and depends on the number of uni-grams in the candidate sentences (c) and the best match lengths for each candidate sentence in the references (r).

$$BLEU = BP \cdot exp(\sum_{n=1}^{N} w_n \cdot \log p_n) \quad (4.1)$$

$$BP = \begin{cases} 1 & \text{if c} > \text{r} \\ exp(1 - \frac{r}{c}) & \text{if c} \leq \text{r} \end{cases} \quad (4.2)$$

Similarly, the Meteor metric was implemented using the NLTK [2] package and implementation of the algorithm. For this algorithm the hypothesis sentence(s) and reference sentence(s) are passed to the function, with the default stemmer [3] of Porter used and default wordnet [4] corpus for the synonym matches.

Meteor leverages the identification of the alignments between the hypothesis and reference by constructing all possible matches between the sentences through exact match, stem match and synonyms where the match shares a membership. The final alignment is the largest subset of all matches, meeting the three criteria with the alignment resolution conducted as a beam search. The final score, see equation 4.3, is calculated through the harmonic mean and applied penalty. The harmonic mean F_{mean} is calculated by placing the majority of the weighting on the uni-gram recall (R) and using the uni-gram precision (P), which is the ratio of the number of uni-grams in the reference translation that are mapped to the total number of uni-grams in the hypothesis translations (Banerjee and Lavie, 2005). The penalty is applied to account for the longer translations that may occur, where chunks are the grouping of mapped uni-grams into the lowest number of chunks such that uni-grams in each chunk are in adjacent positions. Therefore, the longer the n-grams are the fewer chunks are created producing a lower penalty as, if the chunks are equal to 1 the penalty goes to its lower bound determined by the number of uni-grams matched.

[3] https://www.nltk.org/api/nltk.stem.html#nltk.stem.api.StemmerI

[4] https://www.nltk.org/api/nltk.corpus.reader.html#nltk.corpus.reader.wordnet.WordNetCorpusReader

Context	Questions	Answers
Burns was born on July 29, 1953, in Brooklyn, New York, the son of Lyla Smith (nee Tupper) Burns, a biotechnician, and Robert Kyle Burns, at the time a graduate student in cultural anthropology at Columbia University in Manhattan. The documentary filmmaker Ric Burns is his younger brother. Burns' academic family moved frequently. Among places they called home were Saint-Veran, France; Newark, Delaware; and Ann Arbor where his father taught at the University of Michigan. Burns' mother was found to have breast cancer when he was three and she died when he was 11, a circumstance that he said helped shape his career; he credited his father-in-law, a psychologist, with a significant insight: "He told me that my whole work was an attempt to make people long gone come back alive." Well-read as a child, he absorbed the family encyclopedia, preferring history to fiction. Upon receiving an 8 mm film movie camera for his 17th birthday, he shot a documentary about an Ann Arbor factory. He graduated from Pioneer High School in Ann Arbor in 1971. Turning down reduced tuition at the University of Michigan, he attended Hampshire College, an alternative school in Amherst, Massachusetts, where students are graded through narrative evaluations rather than letter grades and where students create self-directed academic concentrations instead of choosing a traditional major. He worked in a record store to pay his tuition. Studying under photographers Jerome Liebling, Elaine Mayes and others, Burns earned his Bachelor of Arts degree in film studies and design in 1975.	Did he attend any other schools?	CANNOTANSWER
Johnny Fean continued to play live music with Stephen Travers, formerly of The Miami Showband. After his retirement, Eamon Carr went on to become a producer of young rock talent in the mid-1980s, and also forming his own record label called Hotwire (which sponsored noted acts such as the punk rock group The Golden Horde). He also did a number of specialist DJ slots on radio before morphing into a music/sports journalist with the Evening Herald in Dublin. More recently he presented on a Dublin station 'Carr's Cocktail Shack' in which he played American music of the 1950s and 1960s. In 2008, Carr and Henry McCullough co-wrote a new bunch of songs. A resulting album entitled Poor Man's Moon was released on 1 September 2008. Also in 2008, Carr released his first book, The Origami Crow, Journey Into Japan, World Cup Summer 2002, a book that is at once a travel log about his journey to Japan, a poetry collection, an homage to Japanese poet Basho, heralded by many as the creator of Haiku, and also has some sports commentary thrown in. Barry Devlin directed for the screen and been a drama writer for radio and screen, as can be seen from his credits on the IMDB and for the radio detective drama Baldi He produced a number of U2 videos in the mid-1980s. Examples of his screen writing are evident in the joint RTE/BBC production Ballykissangel and ITV's The Darling Buds of May. Jim Lockhart is head of production at RTE 2fm and has also done some production work and music arrangement. Charles O'Connor owns two antique shops in Whitby, England. O'Connor continued to record folk and traditional music in his home recording studio.	Are any of the band members retired?	Charles O'Connor owns two antique shops in Whitby, England. O'Connor continued to record folk and traditional music in his home recording studio.
In 1755, Washington became the senior American aide to British General Edward Braddock on the ill-fated Braddock expedition. This was the largest British expedition to the colonies, and was intended to expel the French from the Ohio Country; the first objective was the capture of Fort Duquesne. Washington initially sought an appointment as a major from Braddock, but he agreed to serve as a staff volunteer upon advice that no rank above captain could be given except by London. During the passage of the expedition, Washington fell ill with severe headaches and fever. He recommended to Braddock that the army be split into two divisions when the pace of the troops continued to slow: a primary and more lightly equipped "flying column" offensive which could move at a more rapid pace, to be followed by a more heavily armed reinforcing division. Braddock accepted the recommendation (likely made in a council of war including other officers) and took command of the lead division. In the Battle of the Monongahela, the French and their Indian allies ambushed Braddock's reduced forces and the general was mortally wounded. After suffering devastating casualties, the British panicked and retreated in disarray. Washington rode back and forth across the battlefield, rallying the remnants of the British and Virginian forces into an organized retreat. In the process, he demonstrated bravery and stamina, despite his lingering illness. He had two horses shot from underneath him, while his hat and coat were pierced by several bullets. Two-thirds of the British force of 976 men were killed or wounded in the battle. Washington's conduct in the battle redeemed his reputation among many who had criticized his command in the Battle of Fort Necessity. Washington was not included by the succeeding commander Col. Thomas Dunbar in planning subsequent force movements, whatever responsibility rested on him for the defeat as a result of his recommendation to Braddock.	Are there any other interesting aspects about this article?	He recommended to Braddock that the army be split into two divisions when the pace of the troops continued to slow: a primary and more lightly equipped "

Figure 12: QuAC Dataset Sample

$$Meteor = (1 - pen) \cdot F_{mean} \tag{4.3}$$

$$F_{mean} = \frac{10 \cdot P \cdot R}{R + 9 \cdot P} \tag{4.4}$$

$$Penalty = 0.5 \cdot \frac{\#chunks}{\#uni - grams\ matched})^3 \tag{4.5}$$

Given both metrics were implemented with the NLTK library, it was required to pass the hypothesis or generated text and the ground truth reference text to the function. For both measures, the weighting of the n-grams were even across the number of n-grams, with the calculation being performed at a sentence level for both measures. In performing the metric calculation at the sentence level, the measure was calculated for each translation in the dataset enabling a more granular investigation of the types of translations that were achieving poorer scores. This approach will be discussed within Section 5.

4.4 Implementation

4.4.1 AWS Set-Up

To develop the models outlined, we explored cloud platforms which could host Jupyter notebooks to enable development in Python, chosen for its access to libraries such as Tensorflow and PyTorch. Our development initially began on Google Collaboratory however, RAM limitations of 12GB hindered the training of the models as they can require up to 20GB, this was solved by moving to AWS infrastructure. With the requirement to develop and hypertune models for two datasets over a number of parameters, for BERT we utilized a SageMaker p3.8xlarge instance containing 4 Nvidia GPUs with 64GB of GPU memory and configured an additional 200GB of memory for storage. This instance was used to train all SQuAD and QuAC BERT models for Question Answering. For GPT-2, we employed a p3.16xlarge SageMaker instance containing 8 Nvidia GPUs with 128GB of GPU memory, which had a memory configuration of 50GB for storage. The GPT-2 instance was higher powered due to time constraints for model training and if the work were to be taken up again a smaller instance could be used. This increase in processing power had an increase in cost and significantly decreased the training time of the model, which will be detailed

in later stages.

In order to train the models on SageMaker, Jupyter notebooks were used to develop python code and run command line commands to call python scripts in the case of the BERT models. With the underlying code for the models, both BERT and GPT-2 being of pytorch implementation, the GPUs attached to the instance were automatically detected and the training was split across multiple GPUs. In parallelising the training overhead across multiple GPUs attached to the instances, the time taken for data formatting, tokenizing and training was reduced significantly.

4.4.2 Question Answering

We created Question Answering models for the two datasets of SQuAD and QuAC, by implementing the Hugging Face approach of BERT for Question Answering[5]. During the training phase of the implementation, the model script is passed a number of hyperparameters, training data and evaluation data files, with the evaluation flag set to true to include this stage in the single training run. The training script encompasses the tokenization stage and encodes the data for training and evaluation into the BERT input format including the input masks, input id's and segment masks; this transformation of data is cached this after the initial run to reduce running time for later training instances.

To ensure that the QuAC data could be used with the same BERT model architecture and that we had consistency with the data formats the QuAC data was manipulated to have the SQuAD structure. This process was done before model implementation or training, with the QuAC dataset file and output file name being passed to a python script that performed the reformatting. The script pulled the paragraph data out of the original JSON file, identified the number of questions, answers and answer positions relevant to that paragraph and created the new JSON structure for the paragraph that matched the SQuAD layout; this was then appended to an array. Once each individual paragraph had been appended to the array, the array was assigned to a key in a json and the final json was saved out the new file name. The script for this can be seen in Appendix 8.2. Following this reformatting, the new version of the data was passed as the data files to the script for both training and evaluation.

We undertook an initial training stage to ensure the approach was successful, this was carried out as a testing stage with both datasets and default parameters for both BERT Base and BERT Large models. This initial train achieved poor results and has been excluded from the work but was an indication that the approach worked. From this point a grid search over parameters was employed to determine the best model that could be achieved. The grid search was performed using the sklearn parameter grid library [6] with a python dictionary consisting of keys and arrays of parameters passed to create the grid. Given the research by (Schwager and Solitario, 2019) that indicated parameters of Learning Rate and Epochs could have influence on model outcomes, these two parameters were chosen for the grid search.

Along with these, the batch size being passed to the GPUs for training was also included, as this parameter often affects the generalisation ability of models and can affect training time if the batches are small. The hyperparameters explored are in Table 2 and resulted in 18 models for both datasets trained on different parameter combinations. The warm-up parameter outlined, refers to the percentage of the training data that is used for working up to the learning rate set from 0. This ensures that the model slowly adapts to the new data being trained upon. Although the training loss was not the metric used for determining the best model, the loss graph for the parameters explored for both SQuAD and QuAC can be seen in Figures 13 and 14, showing a steady decrease in the loss in an exponential decay fashion which we can assume that if the curve was extrapolated, the loss would decrease further over more epochs.

During the training the model was saved to checkpoints every 1000 steps and alongside this run information was saved to produce the training graphs through Tensorboard. On training

[5] https://github.com/huggingface/transformers/tree/master/examples/legacy/question-answering

[6] https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.ParameterGrid.html

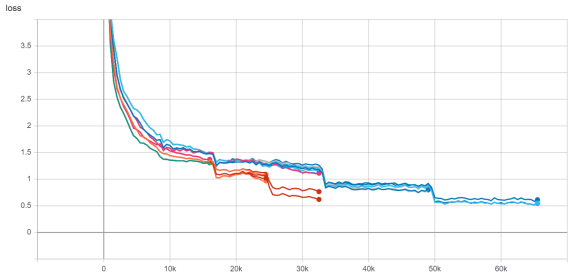

Figure 13: SQuAD QA Training Tensorboard Loss Graph

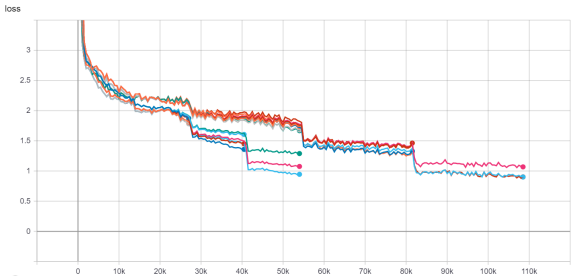

Figure 14: QuAC QA Training Tensorboard Loss Graph

Parameters	Exact Match	F1	Train Time
{"learning_rate": 5e-05, "num_epochs": 2, "per_gpu_batch_size": 4}	65.03	68.83	01:10:46
{ "learning_rate": 3e-05, "num_epochs": 3,"per_gpu_batch_size": 4 }	65.37	69.27	01:45:50
{ "learning_rate": 3e-05, "num_epochs": 2, "per_gpu_batch_size": 2}	68.26	71.60	01:58:43
{ "learning_rate": 2e-05, "num_epochs": 3, "per_gpu_batch_size": 2}	67.53	71.37	03:05:10
{ "learning_rate": 3e-05, "num_epochs": 4, "per_gpu_batch_size": 4}	65.32	69.15	02:17:52
{"learning_rate": 5e-05, "num_epochs": 3, "per_gpu_batch_size": 2}	67.23	71.04	02:57:46
{"learning_rate": 5e-05, "num_epochs": 4, "per_gpu_batch_size": 2}	65.87	69.71	03:55:29
{"learning_rate": 2e-05, "num_epochs": 3, "per_gpu_batch_size": 4}	63.57	67.59	01:41:41
{"learning_rate": 3e-05, "num_epochs": 4, "per_gpu_batch_size": 2}	67.08	70.93	04:06:33
{"learning_rate": 3e-05, "num_epochs": 2, "per_gpu_batch_size": 4}	62.88	66.80	01:11:54
{"learning_rate": 2e-05, "num_epochs": 4, "per_gpu_batch_size": 2}	66.52	70.48	03:49:44
{"learning_rate": 2e-05, "num_epochs": 4, "per_gpu_batch_size": 4}	64.31	68.29	02:13:45
{"learning_rate": 2e-05, "num_epochs": 2, "per_gpu_batch_size": 2}	67.18	70.78	02:00:10
{"learning_rate": 2e-05, "num_epochs": 2, "per_gpu_batch_size": 4}	60.69	64.85	01:09:27
{"learning_rate": 5e-05, "num_epochs": 3, "per_gpu_batch_size": 4}	67.78	71.32	01:43:30
{"learning_rate": 3e-05, "num_epochs": 3,"per_gpu_batch_size": 2}	68.17	71.84	03:02:38
{"learning_rate": 5e-05, "num_epochs": 2, "per_gpu_batch_size": 2}	68.42	71.80	02:00:55
{"learning_rate": 5e-05, "num_epochs": 4, "per_gpu_batch_size": 4}	66.43	70.41	02:19:42

Figure 15: SQuAD Training Metric Results

Parameters	Exact Match	F1	Train Time
{"learning_rate": 2e-05, "num_epochs": 2, "per_gpu_batch_size": 4}	25.85	39.02	01:47:21
{"learning_rate": 2e-05, "num_epochs": 2, "per_gpu_batch_size": 2}	25.44	39.33	03:29:59
{"learning_rate": 5e-05, "num_epochs": 3, "per_gpu_batch_size": 4}	24.10	38.31	03:29:59
{"learning_rate": 3e-05, "num_epochs": 2, "per_gpu_batch_size": 2}	25.56	39.20	03:29:59
{"learning_rate": 5e-05, "num_epochs": 2, "per_gpu_batch_size": 2}	25.78	39.75	03:29:59
{"learning_rate": 3e-05, "num_epochs": 4, "per_gpu_batch_size": 2}	22.50	36.83	05:59:20
{"learning_rate": 2e-05, "num_epochs": 3, "per_gpu_batch_size": 2}	24.30	38.82	03:29:59
{"learning_rate": 3e-05, "num_epochs": 3, "per_gpu_batch_size": 4}	24.16	38.70	03:29:59
{"learning_rate": 5e-05, "num_epochs": 3, "per_gpu_batch_size": 2}	23.52	37.88	03:29:59
{"learning_rate": 2e-05, "num_epochs": 4, "per_gpu_batch_size": 4}	24.08	38.51	03:29:59
{"learning_rate": 2e-05, "num_epochs": 4, "per_gpu_batch_size": 2}	23.47	37.86	06:02:50
{"learning_rate": 3e-05, "num_epochs": 3, "per_gpu_batch_size": 2}	23.88	37.87	04:24:57
{"learning_rate": 3e-05, "num_epochs": 2, "per_gpu_batch_size": 4}	25.44	39.01	03:29:59
{"learning_rate": 2e-05, "num_epochs": 3, "per_gpu_batch_size": 4}	25.02	39.09	03:29:59
{"learning_rate": 5e-05, "num_epochs": 2, "per_gpu_batch_size": 4}	25.05	38.80	03:29:59
{"learning_rate": 3e-05, "num_epochs": 4, "per_gpu_batch_size": 4}	23.20	37.93	03:29:59
{"learning_rate": 5e-05, "num_epochs": 4, "per_gpu_batch_size": 2}	22.82	36.64	05:57:03
{"learning_rate": 5e-05, "num_epochs": 4, "per_gpu_batch_size": 4}	22.84	37.38	03:29:59

Figure 16: QuAC Training Metric Results

initialisation we saved the model parameters and training start time to a text file and on completion the evaluation results and training finish time were added to the file. Training times were dependent on the number of epochs taken with training varying between 1-6 hours, see 15 and 16. During evaluation the overall F1 and EM scores are calculated including how the model performed on those with and without answers; the final model was chosen from the best F1 score across the models. The outcomes for both the SQuAD and QuAC can be seen in Figures 15 and 16, respectively. The best models chosen from this were then taken through to the prediction phase of the project to evaluate their performance further; these being the models with parameters {"learning_rate": 3e-05, "num_epochs": 3, "per_gpu_batch_size": 2} for SQuAD and {"learning_rate": 5e-05, "num_epochs": 2, "per_gpu_batch_size": 2} for QuAC.

It is clear from Figure 16, that the models for the QuAC dataset did not train and perform as strongly as those with the SQuAD dataset. In order to improve the models performance, a number of further training tests were carried out by reducing and increasing the stride length through the context paragraph. This was motivated by the fact that QuAC paragraphs are on average longer than SQuAD, leading to potentially missing the answer within the text. The process did not improve the models performance. Additionally, changes in the batch size by increments of two in the range 8-16 and in the number of epochs in increments of two in the range 12-20, were also considered. Similar to the previous outcome we achieved no increase in model performance. In researching similar papers it was discovered that BERT models without additional layers to handle historical questions or context performed similarly poorly to the results we were seeing.

In reading the work by (Sen and Saffari, 2020) and (Qu et al., 2019), where an F1 of 33.3 and 62.4 were respectively achieved, it was determined that our poor results were to be expected. Despite our QuAC models poor performance the best model was taken forward to the further evaluation stage. Table 3 summarises how our best models from the training stage compare to those from leading papers. The SQuAD model developed is only slightly below (6%) that of a leading paper, with the main difference being the batch size we used in comparison; ours being 2 or 4 and theirs being 24.

With the trained model developed we are able to perform answer predictions on provided context

	SQuAD	QuAC
Epochs	[2, 3, 4]	[2, 3, 4]
Batch Size	[2, 4]	[2, 4]
Learning Rate	[2e-5, 3e-5, 5e-5]	[2e-5, 3e-5, 5e-5]
Warm Up	0.1	0.1

Table 2: QA Model Initial Parameters

paragraphs. This is done for the evaluation set of both the SQuAD and QuAC dataset, through their trained models respectively. To perform answering we read the evaluation data into a dataframe with columns of context, question, answer and answer start; by having the ground truth answer in the dataframe we are able to measure the model performance in later evaluations. For each row in the dataframe the context and question are tokenized and encoded into the BERT input format including the [CLS] and [SEP] tokens. These encoded inputs are then passed to the trained QA model to determine the answer span start and end positions; these values are then used to identify the location of the answer in the context and return the answer in word form. Once all the answers have been obtained for the dataframe rows, they are added into a new column and the final results are saved out to a csv file. In future it is possible that these answers would be outputted in real time to a front end interface after receiving the context and question generated by the QG model.

4.4.3 Question Generation

Similarly to the Question Answering models, two Question Generation models were developed from the two datasets SQuAD and QuAC. We implemented these through the library GPT-2 simple[7]. This package wraps the original scripts from OpenAI for the small and medium GPT-2 model versions into a python package. This approach was chosen due to the ease of implementation and readily available documentation, however with the library wrapping the original scrips, it is possible to replace this section with either the original or Hugging Face implementation of GPT-2. The package encompasses the tokenizing, finetuning/training and generation stages of the GPT-2 model. Due to size constraints in training the model, the small 124M parameter GPT-2 model was chosen for the Question Generation models.

In order to use the json datasets with the GPT-2 model, we reformatted the data into the preferred text file format and in doing this we included custom tokens to aid the understanding of context and questions within the model. Firstly the json files were read and converted into python lists of context paragraphs, questions, answers and answer starts using a function created for previous QA formatting. For GPT-2 models we are only interested in the context and questions meaning these were the only two lists used. The context paragraph and questions list were passed into a function that cleaned the newline characters from the context paragraphs and created a dataframe of two columns from the two lists. The dataframe was then accessed per row to generate a string in the format:

$$\begin{aligned} &< |startoftext| > [CONTEXT] : \\ &context\ string\ [QUESTION] : \\ &question\ string < |endoftext| > \end{aligned} \tag{4.6}$$

that was then added to a new column in the dataframe; this results in duplicated contexts paired with varying questions. Once all the formatted strings had been acquired, the rows were joined together into one string and written out to the text file. This process was performed for both SQuAD and QuAC, training and evaluation datasets. For the evaluation datasets, duplicate context paragraphs were dropped from the dataframe, as we generate 5 questions per context in the generation stage which would result in many duplicate questions over the duplicated paragraphs.

When the modified data files have been obtained the file name is passed as a parameter to the finetune function alongside other hyperparameters. In doing this, the finetune function subsequently calls the encoding functions that transform the text into a format consumable by GPT-2. GPT-2 inputs data that is in the BPE (Byte Pair Encoding) format which is performed invoking the vocabulary file

[7] https://github.com/minimaxir/gpt-2-simple/tree/afe38c43c8b40d1e81d0a57ccfa1a3138e7626a0

provided with the pre-trained GPT-2 model. This encoding technique reduces the words in the dataset into smaller representations by identifying patterns of characters and setting their corresponding representation to a character, or byte, that is not used in the data. Once the data has been encoded we can use this representation to train the model.

For GPT-2 there are a number of parameters that could have an effect on the performance of the trained model and therefore, hypertuning was performed over these parameters to find the best model for generation. We used the approach of Grid Search with the Parameter Grid function from SkLearn [6] to iterate over a combination of parameters of batch size, learning rate and steps. The optimiser Adam was included however, this remained constant over the grid search and was chosen over the SGD optimizer as it often has a better performance. The parameters passed to the model can be seen in Table 4. With the default and recommended batch size and learning rate for GPT-2 being 8 and $1e^{-}5$ respectively, the hyperparameters chosen were around these values to ensure that the model was optimized for the data being used.

	SQuAD	**QuAC**
Batch Size	[6, 8]	[6, 8]
Optimizer	['Adam']	['Adam']
Learning Rate	[1e-4, 1e-5, 1e-6]	[1e-4, 1e-5, 1e-6]
Steps	[2000, 4000]	[2000, 4000]

Table 4: QG Model Hyper-Parameters

The Tensorboard training loss graphs for both the SQuAD and QuAC GPT-2 models can be seen in Figures 17 and 18. In analysing these training graphs, we determined that the number of training steps and the learning rate had the most impact on the model loss decreasing. The training times of the GPT-2 models were highly dependent on the number of steps being taken, with the 2000 step models taking 20 minutes and the 40000 step models taking 40 minutes. However, it is worth noting the training time is also dependent on computational resources and our short training times were due to having access to 8 Nvidia GPU's. With machine generated text we had no way to evaluate the generated data against a ground truth where we could take an F1 scores approach to measure the model performance, as the text generated may vary from the ground truth but still make sense lexically. Therefore to determine the model with the best performance for QG, we take the model with the lowest loss through to the generation stage. This can be judged visually from the loss graphs or can be determined from the loss table generated from the Tensorflow event run files, seen in Figures 19 and 20 . When looking at the loss graphs it is evident that in extrapolating the curves there is the possibility of the loss decreasing further. Therefore, a greater number steps could be taken during training with early stop methods implemented when a target loss value is reached to prevent over-fitting and decrease the loss.

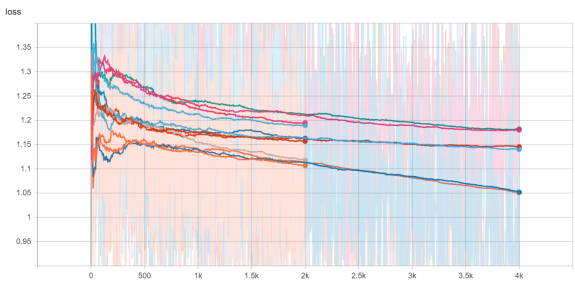

Figure 17: SQuAD QG Training Tensorboard Loss Graph

Once we have the trained models for both

	SQuAD	**SQuAD (Schwager and Solitario, 2019)**	**QuAC**	**QuAC (Sen and Saffari, 2020)**
Exact Match	68.171	73.85	25.78	NA
F1	71.844	76.70	39.74	33.3

Table 3: Best Model Results Compared to Leading Papers

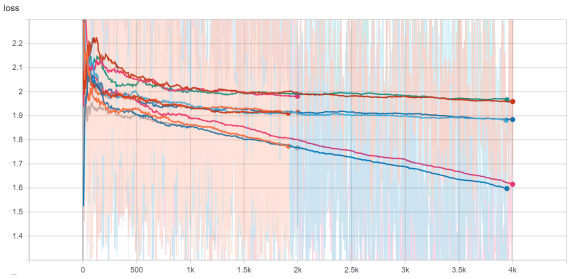

Figure 18: QuAC QG Training Tensorboard Loss Graph

Run	Parameters	Loss
0	{'batch_size': 6, 'learning_rate': 0.0001, 'optimizer': 'adam', 'steps': 2000}	0.93
1	{'batch_size': 6, 'learning_rate': 0.0001, 'optimizer': 'adam', 'steps': 4000}	0.95
2	{'batch_size': 6, 'learning_rate': 1e-05, 'optimizer': 'adam', 'steps': 2000}	1.02
3	{'batch_size': 6, 'learning_rate': 1e-05, 'optimizer': 'adam', 'steps': 4000}	1.34
4	{'batch_size': 6, 'learning_rate': 1e-06, 'optimizer': 'adam', 'steps': 2000}	1.27
5	{'batch_size': 6, 'learning_rate': 1e-06, 'optimizer': 'adam', 'steps': 4000}	1.29
6	{'batch_size': 8, 'learning_rate': 0.0001, 'optimizer': 'adam', 'steps': 2000}	1.16
7	{'batch_size': 8, 'learning_rate': 0.0001, 'optimizer': 'adam', 'steps': 4000}	1.06
8	{'batch_size': 8, 'learning_rate': 1e-05, 'optimizer': 'adam', 'steps': 2000}	1.15
9	{'batch_size': 8, 'learning_rate': 1e-05, 'optimizer': 'adam', 'steps': 4000}	1.11
10	{'batch_size': 8, 'learning_rate': 1e-06, 'optimizer': 'adam', 'steps': 2000}	0.95
11	{'batch_size': 8, 'learning_rate': 1e-06, 'optimizer': 'adam', 'steps': 4000}	1.20

Figure 19: SQuAD QG Training Metric Results

SQuAD and QuAC datasets, it is possible to load these in from their saved model files to perform generation on evaluation data. For this task the models are given the evaluation text files that have no duplicate context paragraphs, as for each paragraph we will be generating multiple questions to increase variation and to reduce the risk of duplicate generated questions. We start by loading the evaluation data into a dataframe as we pre-process the text to remove the ground truth questions from the text snippet for the model to generate text after the [QUESTION]: token. The input format of the text for the model can be seen in 4.7. The formatted context entries are then passed one at a time to the generate function from GPT-2 Simple [7], that uses the loaded fintuned model to perform the generation. During the generation there are a number of parameters that can be set, we looked at temperature, batches, number of samples and set return as list to True, to have our generated text returned from the generate function. The batches and number of samples parameters are tuned depending on the number of samples you wish to generate, we wanted 5 questions per context and therefore set these both to 5. The more samples that are generated the greater possibility there is that the model will repeat the text generated, therefore we kept the generated questions small to reduce this and also due to the large number of contexts we had. The temperature parameter controls the creativity of

Run	Parameters	Loss
0	{'batch_size': 6, 'learning_rate': 0.0001, 'optimizer': 'adam', 'steps': 2000}	1.96
1	{'batch_size': 6, 'learning_rate': 0.0001, 'optimizer': 'adam', 'steps': 4000}	1.18
2	{'batch_size': 6, 'learning_rate': 1e-05, 'optimizer': 'adam', 'steps': 2000}	2.01
3	{'batch_size': 6, 'learning_rate': 1e-05, 'optimizer': 'adam', 'steps': 4000}	1.55
4	{'batch_size': 6, 'learning_rate': 1e-06, 'optimizer': 'adam', 'steps': 2000}	2.18
5	{'batch_size': 6, 'learning_rate': 1e-06, 'optimizer': 'adam', 'steps': 4000}	1.71
6	{'batch_size': 8, 'learning_rate': 0.0001, 'optimizer': 'adam', 'steps': 2000}	1.55
7	{'batch_size': 8, 'learning_rate': 0.0001, 'optimizer': 'adam', 'steps': 4000}	1.11
8	{'batch_size': 8, 'learning_rate': 1e-05, 'optimizer': 'adam', 'steps': 2000}	2.13
9	{'batch_size': 8, 'learning_rate': 1e-05, 'optimizer': 'adam', 'steps': 4000}	1.70
10	{batch_size': 8, 'learning_rate': 1e-06, 'optimizer': 'adam'', 'steps': 2000}	2.05
11	{'batch_size': 8, 'learning_rate': 1e-06, 'optimizer': 'adam', 'steps': 4000}	2.46

Figure 20: QuAC QG Training Metric Results

the model and ranges between 0 and 1, with 0 producing the same generated text each time and as we tend to 1 we generate more creative text. In order to test the model and later test the ability of the QA models, we chose a temperature of 0.7 to produce fairly creative questions; the range of 0.7-1 is recommended in GPT-2 simple documentation.

The 5 questions generated for each context are returned as a list, which are expanded out into 5 rows of context and a single question. This final data representation is saved out to a csv file that is used with the QA model, where the generated questions and context paragraphs for the SQuAD or QuAC datasets are passed to their matching QA model. At this stage the context and question are encoded and passed to the QA model, as outlined in section 4.4.2, and the output answer predictions are added to the dataframe that is saved to an output csv for evaluation. In a pipeline implementation, the 5 questions and context would be passed to the QA model for answering in real time.

$$< |startoftext| > [CONTEXT] : \\ context\ string\ [QUESTION] : \tag{4.7}$$

4.4.4 Pipeline

With both the QA and QG models trained across both datasets and the optimum models chosen based on either loss or F1 scores, we can address the linkage of the two. Shown in Figure 21 is our architecture for how the QA and QG models link together. The QG GPT-2 model comes first in the pipeline and takes in the context paragraph, from this we then generate a number of questions. These questions and the context paragraph are then processed through the QA BERT model to receive the answers. In using this approach, the evaluation

technique and metrics for the QA model are able to be used to evaluate the QG model. As we can say that if the question is able to be answered to a high standard, we can thus determine the question itself was of a high quality. This will be explored further in the following evaluation section.

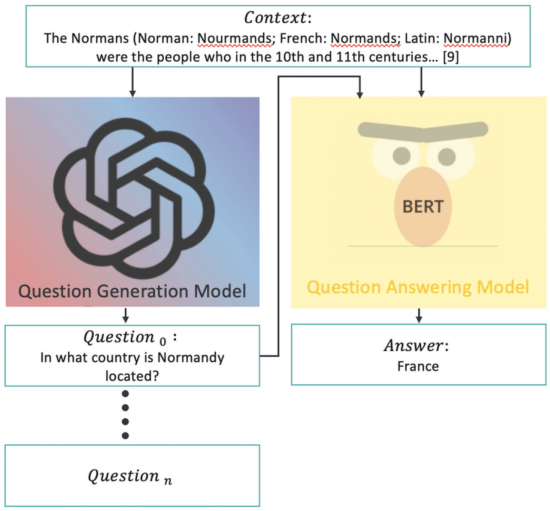

Figure 21: Model Pipeline

5 Evaluation

5.1 Evaluation Process

We evaluate all 4 models, that being the QA and QG models for both SQuAD and QuAC by using the evaluation csv files outlined in our implementation. For QA these csv files contain the ground truth data of context, question, answer and answer start and the prediction of the answer by the models. For QG the csv contains the context and generated questions, however when we read this file we add a third column containing a list of the ground truth questions corresponding to each context paragraph for each row in the dataframe.

With the evaluation csv files read into a dataframe we use the measures of BLEU and Meteor to analyse the machine translated text produced from either the QA or QG model. The metrics of BLEU and Meteor are calculated per sentence in the dataframe rather than over the corpus, as this enables the understanding of subsets that may perform poorly. To calculate the BLEU score we pass the original ground truth text, be that the answer or list of questions, and the hypothesis sentence, being the answer or the generated question, depending on the model being evaluated. The sentences being compared are firstly converted to lowercase, as we know that this should have no difference on the measure of their match, then we look at the length of the sentences, where sentences of different lengths they are made to be the same size with blank spaces, as the algorithm can only handle texts of the same length. The BLEU function is then called for each n-gram type, 1 through to 4, with equal weights over the n-grams being considered and a smoothing method of version 4 applied to reduce the bias towards shorter translations. We then pass the 4 lists of BLEU scores back to the dataframe and create 4 columns to hold the values. The Meteor score is calculated in the same function call, which is able to be calculated without the length or lowercase steps required for BLEU and utilizes the meteor_score function from NLTK to calculate the score. The Meteor scores for the dataframe rows are added to a list which then populates a column in the evaluation dataframe; a sample of this evaluation dataframe can be seen in Figure 22. In producing these metric columns it was noticed that for SQuAD datasets the BLEU 1 score provided a more accurate picture against human judgement and was closer to the Meteor score; however, for QuAC this was shown through the BLEU 4 score. This matches the understanding that for SQuAD shorter more factual answers or questions and generated whereas for QuAC more context based questions and answers are considered which can be lengthier. Therefore, for the two datasets we looked at the different BLEU scores alongside the Meteor score to evaluate the models; BLEU 1 for SQuAD and BLEU 4 for QUAC.

Additionally, an analysis on the type of questions being asked was performed. We tagged each question by considering the 5 W-questions (Who, What, Where, When and Why) as well as How, Which and Undetermined. This exercise was conducted by creating a dictionary of the question types where the keys are the types and values are unique numbers 0-7, and iterating over the rows in the questions column to search if the question string contains one of the types. The Undetermined type was used if the question did not contain any of the prescriptive question types. This tagging information is set to be a further two columns in the dataframe, with an example of the table shown in Figure 22. The addition of the question

Context	Question	Answer	Answer Text	Answer Pred	BLEU1	BLEU2	BLEU3	BLEU4	Meteor	Q Types	Type Code	Vectors	Reduced Vector
Warsaw's mixture of architectural styles reflects the turbulent history of the city and country. During the Second World War, Warsaw was razed to the ground by bombing raids and planned destruction. After liberation, rebuilding began as in other cities of the communist-ruled PRL. Most of the historical buildings were thoroughly reconstructed. However, some of the buildings from the 19th century that had been preserved in reasonably reconstructible form were nonetheless eradicated in the 1950s and 1960s (e.g. Leopold Kronenberg Palace). Mass residential blocks were erected, with basic design typical of Eastern bloc countries.	When was Warsaw completely razed to the ground by bombing raids?	97	During the Second World War	second world war	0.06	0.15	0.18	0.19	0.61	When	3	[2043, 2001, 8199, 3294, 10958, 5422, 2000, 1996, 2598, 2011, 8647, 11217, 1029]	[983.1297646454813, -1141.3226225545486]
The University is organized into eleven separate academic units,Âîten faculties and the Radcliffe Institute for Advanced Study,Âîwith campuses throughout the Boston metropolitan area: its 209-acre (85 ha) main campus is centered on Harvard Yard in Cambridge, approximately 3 miles (5 km) northwest of Boston; the business school and athletics facilities, including Harvard Stadium, are located across the Charles River in the Allston neighborhood of Boston and the medical, dental, and public health schools are in the Longwood Medical Area. Harvard's $37.6 billion financial endowment is the largest of any academic institution.	How many academic units make up the school?	33	eleven separate academic units	eleven	0.44	0.37	0.32	0.28	0.14	How	6	[2129, 2116, 3834, 3197, 2191, 2039, 1996, 2082, 1029]	[-7042.871183592445, -3873.926199998317]
In the U.S. federal health care system (including the VA, the Indian Health Service, and NIH) ambulatory care pharmacists are given full independent prescribing authority. In some states such North Carolina and New Mexico these pharmacist clinicians are given collaborative prescriptive and diagnostic authority. In 2011 the board of Pharmaceutical Specialties approved ambulatory care pharmacy practice as a separate board certification. The official designation for pharmacists who pass the ambulatory care pharmacy specialty certification exam will be Board Certified Ambulatory Care Pharmacist and these pharmacists will carry the initials BCACP.	What entities are included in the federal health care system?	50	the VA, the Indian Health Service, and NIH	va , the indian health service , and nih	0.10	0.20	0.23	0.22	0.60	What	1	[2054, 11422, 2024, 2443, 1999, 1996, 2976, 2740, 2729, 2291, 1029]	[-4250.987551196401, -3869.930834769487]

Figure 22: Sample of Evaluation Table

types enables us to later categorise the poorly performing data by colour within plots for better visualisation, by assigning each type a unique colour.

Alongside the columns for the evaluation data, metric and question tagging additions, we reformat the question data in order to create plots and visualisations. In doing so we can analyse the questions, as the aim is to understand the subset of questions that may produce poor results. With the questions being text data the first step performed was to encode these into their vector representations, which was completed with a pre-trained BERT tokenizer. These vectors obtained were of many dimensions and in this state, are unable to be plotted in a recognisable space for the human eye. Due to this, Principal Component Analysis (PCA) (Wold et al., 1987) was performed on the question vectors to reduce them into a two dimensional representation while retaining the original information. As a further goal we would analyse the number of components required to represent a suitable percentage of the information and wisely choose the number of dimensions kept during reduction. The vector reduction provides a 2D representation of the question that can then be used to plot the question in a 2D graph.

5.2 QA Evaluation

With the evaluation dataframe for both the SQuAD and QuAC QA BERT models, we analysed the thresholds for the BLEU and Meteor metrics to determine at what point the answers became of poor quality. Here we defined poor quality as both a score below the threshold set for the metrics and where there is no answer found, which can be shown by a blank space or the [CLS] token. We rely more heavily on the Meteor score to determine the answers quality, as the BLEU metric, is less robust in comparison. This is due to the finding that a low BLEU score did not necessarily correspond to a poor answer, as the Meteor score can be higher and the answer found can make sense given the context and question. We explored a variation of threshold values to determine the poor answers, through manual inspection across values in the range 0.1-0.4; on examining the results in the threshold table it was deemed that 0.2 was a sufficient value to segregate the data. With the threshold value set, the subsets are determined by looking at both the BLEU and Meteor measures and using and logic, where if the BLEU and Meteor scores are both less than 0.2 they are deemed as poor answers.

On obtaining the poor answer subsets for both the SQuAD and QuAC datasets, we firstly analysed the amount of questions that are answered correctly and incorrectly in the within their question sub-types. The donut pie charts for both SQuAD and QuAC can be seen in Figures 23 and 24, which show the amount in each question type on the outer ring and the breakdown of correct and incorrect on in the inner ring. Looking at Figure 23, it is clear that the QA model performs relatively strongly across all question types, achieving a correct majority in each type. It is clear that the data has a high number of What questions compared to other categories, which could provide a skew to the model however, from the plot it appears this skew has had a minimal effect. This could be addressed in future variations by sampling the dataset for training more evenly across the question types. Compared to the SQuAD model, the QuAC model performs far worse, however this is to be expected due to the poor F1 score for the QuAC QA model. In analysing the donut plot in Figure 24, we can see that the incorrect to correct ratio is roughly even, showing that the model not only has a poor

performance but is not strong on any one category. Within this dataset we have a large number of Undetermined question types, which is due to the context questions present in the data such as those starting with Did and Are.

To understand the breakdown of correct and incorrect questions across the question types, we analysed if there were commonalities amongst the questions being answered poorly and marked as incorrect. This was done through scatter plot visualisations shown in Figures 25 and 26, which were generated using the reduced vector representations of the question text. The two entries in the reduced vector are used as the x and y coordinates, and the colouring of the data points is performed using the question types. Looking at the SQuAD scatter plot, it is clear we have a large amount of What question type points that have a large spread across the axis, this could indicate that these questions could be more unique than those seen previously. Within the QuAC scatter plot it is clear that there are significantly more data points, with the majority being of What, How and Undetermined types, correlating with the donut plot where these are the most represented question types. However, a large number of data points clustered in a single graph area results in overlapping which obscures the insights of possible clusters. With a large clustering group in the bottom left of both SQuAD and QuAC graphs, it is clear that the incorrect questions are largely similar in their reduced vector composition. This could be explored further by increasing the dimensionality into a 3D space to determine if more defined clusters could be identified.

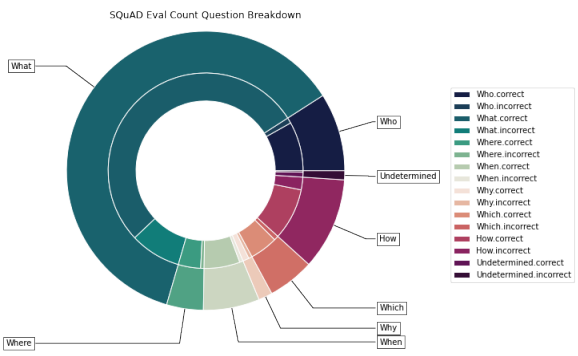

Figure 23: SQuAD QA Donut Plot

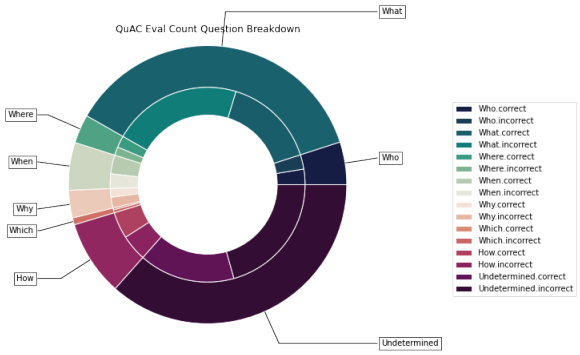

Figure 24: QuAC QA Donut Plot

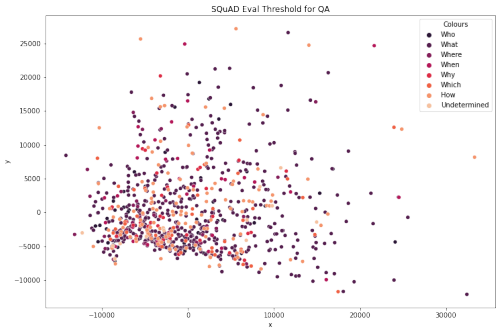

Figure 25: SQuAD QA Scatter Plot

5.3 QG Evaluation

A similar approach as outlined above is taken to evaluate the SQuAD and QuAC QG models, with an additional stage added to ensure generated question quality. The evaluation dataset for the two QG models contains the BLEU and Meteor scores for the generated question against the ground truth questions however, we also deem a good question as one that is able to be answered. Therefore, after we use the threshold of 0.2 on both the BLEU and Meteor metrics, taking the same *and* logic, we also check that the question was answered by considering the answer prediction from the matching QA model. The questions without answers are considered poor questions, as these show that the QA model was not able to answer the generated question and that the generation was of a low score meaning that this is a generation model failure. The questions below the threshold which are answered are deemed to be poor questions with the caveat that the QA model was able to attempt an answer regardless. While the metric scores cannot be a full measure as there is no guarantee that the generated questions will closely match the ground truth questions, the

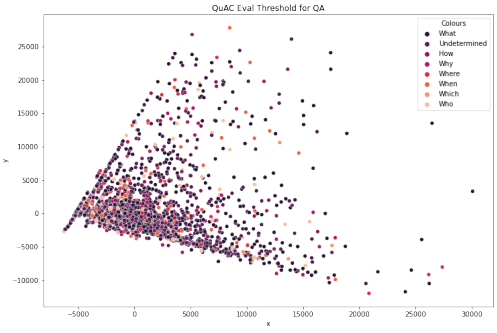

Figure 26: QuAC QA Scatter Plot

metrics tell us how close the generated questions are to the ground truth and if they make sense from a language standpoint.

With the QG models we analyse the question types of the generated questions to understand the ratio of question types generated and the ability of the QA model to answer the generated questions, per question type. The donut breakdown plots for both the SQuAD and QuAC generated questions are shown in Figures 27 and 28. The interest of note in both donut plots when compared to the question types from the QA evaluation, is that there are significantly more What type questions. This shows that while the dataset skew towards this question type did not overly affect the QA models ability to answer, it did affect the number of questions that the QG models generated within question types. Along with this there has been a reduction of Undetermined question types being generated from the QG models.

Looking at the SQuAD generated questions donut plot, a strong performance of the questions through the QA model is evident as a large proportion in each question type are correct; similar to previous evaluation data results. This proves that the QA model is robust enough to handle the generated questions. This cannot be said for the QuAC donut plot, as the QA model is performing poorly on the generated questions, with an even ratio of correct to incorrect similar to what was seen before in the QA evaluation donut. This outcome could be due to either the generated questions being too difficult or that the QA model has a poor performance, in this case it is believed to be the latter. Therefore it can be said that the questions generated were of a similar quality to those within the QuAC dataset but that the QA model was not able to handle these and answer effectively.

To analyse the types of questions being generated that were of a poorer quality, i.e. under the threshold value 0.2, and understand the those that are unanswerable, we used two scatter plots. The first scatter displays all of the data points that were below the threshold and marked as incorrect, Figures 29 and 31, while the second plots, Figures 30 and 32, display the data points that were below the threshold that were not answered. The SQuAD plot with all data points below the threshold, regardless of being answered or not, have a high number of poor What questions which correlates to this having a large number of questions generated in this type. In removing the poor questions that were answered, we get Figure 30 containing 42% of the poor questions. From this we determined the unanswered data points are the majority of the outlier points, which represents variation in the text within a questions construction and that the majority of points are of the What type. The tight cluster indicates that there are a number of similar questions being generated of a poor quality, that would be down to similar wording or repetition in the generation. The QuAC scatter plot for the threshold questions contains 42.8% of the total generation questions which are overlapping in one main cluster with a few outliers. Similar to SQuAD the What type is most prevalent as this is also the majority question type. In comparing this to the second scatter plot with answered questions removed, Figure 32, it is clear that the QA model handles most of the outliers and the problem area is the tight cluster which depicts questions of a similar nature are either handled poorly by the QA model or they are poorly generated.

The Table 5 outlines the summary statistics for the evaluation stages for both QA and QG on the two datasets. The fifth row indicates an experiment with the QuAC generated questions. As the QuAC QA model had a poor performance, the generated questions were ran through the SQuAD QA model to determine if its knowledge would transfer to a new set of questions. The Unanswered column is of the most interest, as for the QuAC QA model with the generated questions this was 48% but for the SQuAD QA model there was an increase in

Type	Average BLEU	Average Meteor	Num below threshold	Below Threshold	Unanswered	Unanswered below threshold
SQuAD Eval	0.035	0.024	839	14.15%	10.88%	7.27%
QuAC Eval	0.038	0.021	4042	54.96%	53.64%	35.90%
SQuAD Gen through SQuAD QA	0.152	0.098	1020	18.15%	41.11%	7.93%
QuAC Gen through QuAC QA	0.176	0.088	394	42.87%	47.99%	21.87%
QuAC Gen through SQuAD QA	0.176	0.088	394	42.87%	25.90%	12.19%

Table 5: Evaluation Summary Statistics

performance with only 25% of questions being unanswered. The increase in questions being answered leads us to conclude that the SQuAD QA model is capable of being used on other question answering datasets. This column also indicates that there are a high percentage of the generated questions being left unanswered for both SQuAD and QuAC, leading to the conclusion that the generation may be forming too creative questions for the QA models to handle.

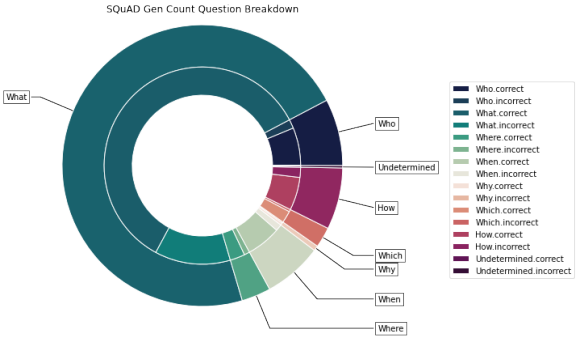

Figure 27: SQuAD QG Donut Plot

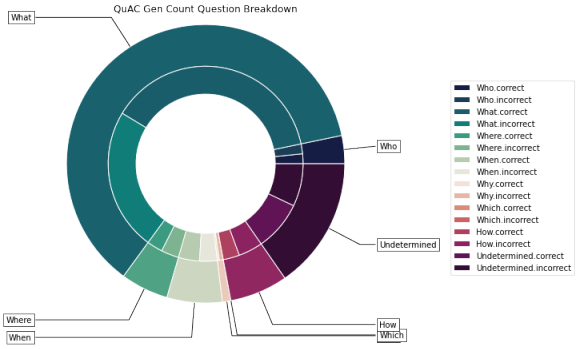

Figure 28: QuAC QG Donut Plot

6 Conclusion

In this work we outlined the development of a QG to QA pipeline trained across two datasets, to demonstrate the linkage between both models and the impact of training datasets on model performance. These were implemented using Pytorch and python packages to develop two language models of BERT and GPT-2 for QA and QG respectively. Both models were trained using a hypertuning approach of grid search, to find optimum parameters for model performance over SQuAD and QuAC datasets. Within the QA models, we achieved a strong performance on questions that required factual answers and less context dependant questions. A 71.84 F1 score was achieved for SQuAD while a poor 39.74 F1 score was achieved on QuAC, which highlights the different nature of the data present in the two datasets. For QG it was observed that more creative questions could be handled by the QA models, to a certain degree and that the QuAC dataset with more context dependent text, resulted in more undetermined question types. We analysed the subsets of incorrect or poor answers for the evaluation set and generated questions. Our results showed that the QA model was less affected in the training stages by a skew in the question types within the training data, such as What type questions. However, the QG model would generate more questions of the types that were

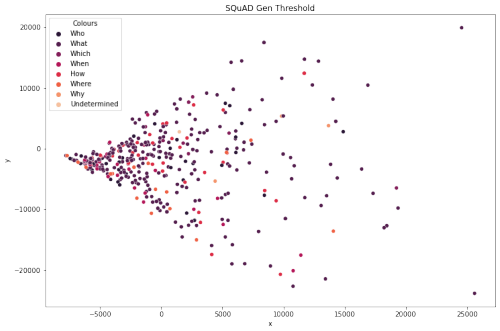

Figure 29: SQuAD QG Scatter Plot (all data points)

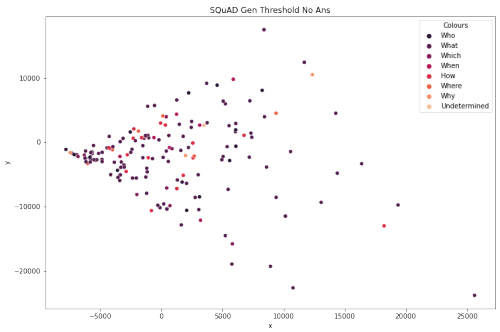

Figure 30: SQuAD QG Scatter Plot (no answers)

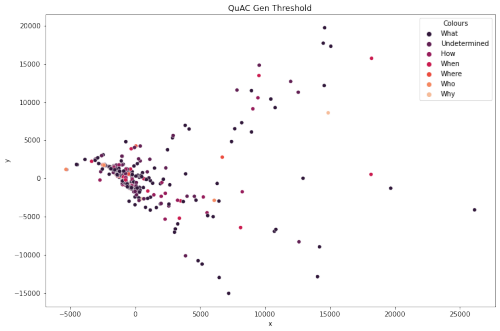

Figure 31: QUAC QG Scatter Plot (all data points)

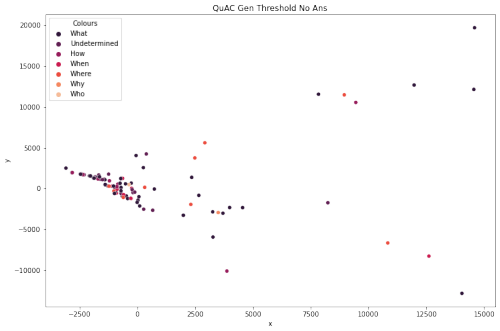

Figure 32: QUAC QG Scatter Plot (no answers)

a majority presence. Further to this we explored the quality of the questions and answers produced using BLEU and Meteor metrics, which were utilized to threshold the questions into subsets that were answered poorly by the QA models. In performing these measures it was demonstrated that the SQuAD model performed significantly better than QuAC, with 10.88% and 53.64% of questions left unanswered respectively. While the implementation of the BERT and GPT-2 models is not unique to this space, the novel value in this work is the evaluation of question types and the subsets of questions that cause poor performance.

Based on the current implementation and work completed, there are a number of areas we have identified as future work and developments. The default optimizer used of AdamW or Adam for both BERT and GPT-2, will be experimented upon with replacements such as LAMB (You et al., 2020) and LARS (You et al., 2017). Further to this, an improvement of parameter tuning will be achieved through a Baysian Optimisation approach and in expanding the range of parameters to be hypertuned; for example the temperature required for GPT-2 generation to investigate the creativity of generated text.

The training data chosen presented concerns, as a skew in the dataset towards certain types of questions affected the QG model and resulted a larger skew being present within the generated set. Further exploratory data analysis would be conducted to balance the question types represented across the considered datasets. The combination of multiple datasets as part of the model training will also be explored. The aim will be to investigate the impact that finetuning over different datasets has on the models ability to retain the knowledge from earlier data, to strengthen their ability on new datasets. For example a QA or QG model finetuned with SQuAD, then QuAC and tested on unseen data from both. This development is motivated by the test in Section 5 where the SQuAD QA model was able to answer reasonably for the QuAC generated questions.

To improve the performance of the QuAC QA

model, further work into the structure of BERT and additions to aid the understanding of context will be undertaken. Taking inspiration from (Qu et al., 2019), an approach of adding additional layers to the BERT base model, to increase the performance over context based datasets will be investigated. Similarly the architecture of the GPT-2 model will be explored, through an increase in the number of decoder layers within the Transformer to increase performance. Coupling QA and QG model experiments, the impact of feeding the loss from the QA model to the QG model will be explored to improve the generated questions (Klein and Nabi, 2019).

Within the evaluation, the approach of PCA to reduce the question vectors to 2D vectors and the possible loss of information during reduction will be explored. The number of principal components required to accurately represent a large percentage of the question information will be investigated, alongside the number of dimensions for cluster visualisations. Further to this, the number of classes for the questions types will be expanded to include context driven types such as Did and Are. These are prominent in context dependent questions and would reduce the number in the Undetermined group for more accurate evaluation. Lastly, a similarity based model will be developed to group the vectors of the question text, for a more precise measure of similarity, as the question vectors will not be reduced into lower dimensions causing a risk of information loss.

Finally the creation of a responsive user interface will be implemented, to make the QA and QG models usable and consumable outside of a coding environment. This would be achieved by developing an interface connected to model endpoints. To implement this, a responsive front-end would send requests for predictions of questions or answers to the model endpoints within AWS SageMaker, with the prediction response populating the interface when returned.

This work illustrates how QG and QA can be implemented in a pipeline system. This implementation has the ability to enhance systems that require conversational AI, such as chatbots, which often use logic driven behaviour and preformatted responses. An example of this is where documents or textual information is stored in an unstructured format that requires querying based on customer interaction. The QA model would have the textual information for the customer and their question, to provide an answer in the response through a chatbot interface. The pipeline of both models could also be used in a teacher-student scenario, where the textual information is provided to the QG model to generate a set of questions, that can test a student's knowledge. The answers would then be predicted to enable an answer for the student, to benchmark their knowledge against. The development of models and pipelines such as QA to QG, are fundamental for conversational AI systems and model understanding.

To summarise, in this paper we develop and demonstrate the capability of a QG to QA pipeline to be developed with BERT and GPT-2 model architectures and outline a number of improvements that would enable an increase in performance to be achieved. Our evaluation highlights the models strengths and weaknesses within question types and the models developed could be easily deployed to achieve the pipeline via model endpoints and a front-end interface.

7 Acknowledgements

I would like to acknowledge the support and guidance over the year of my Queen's University Belfast supervisor Dr Barry Devereux and my Aflac Northern Ireland supervisor Dr Joana Cavadas. Alongside this I would also like to acknowledge Aflac Northern Ireland, for both their collaboration on this project and the access to compute resources required to train and evalute the developed models.

8 Appendix

8.1 Gantt Chart

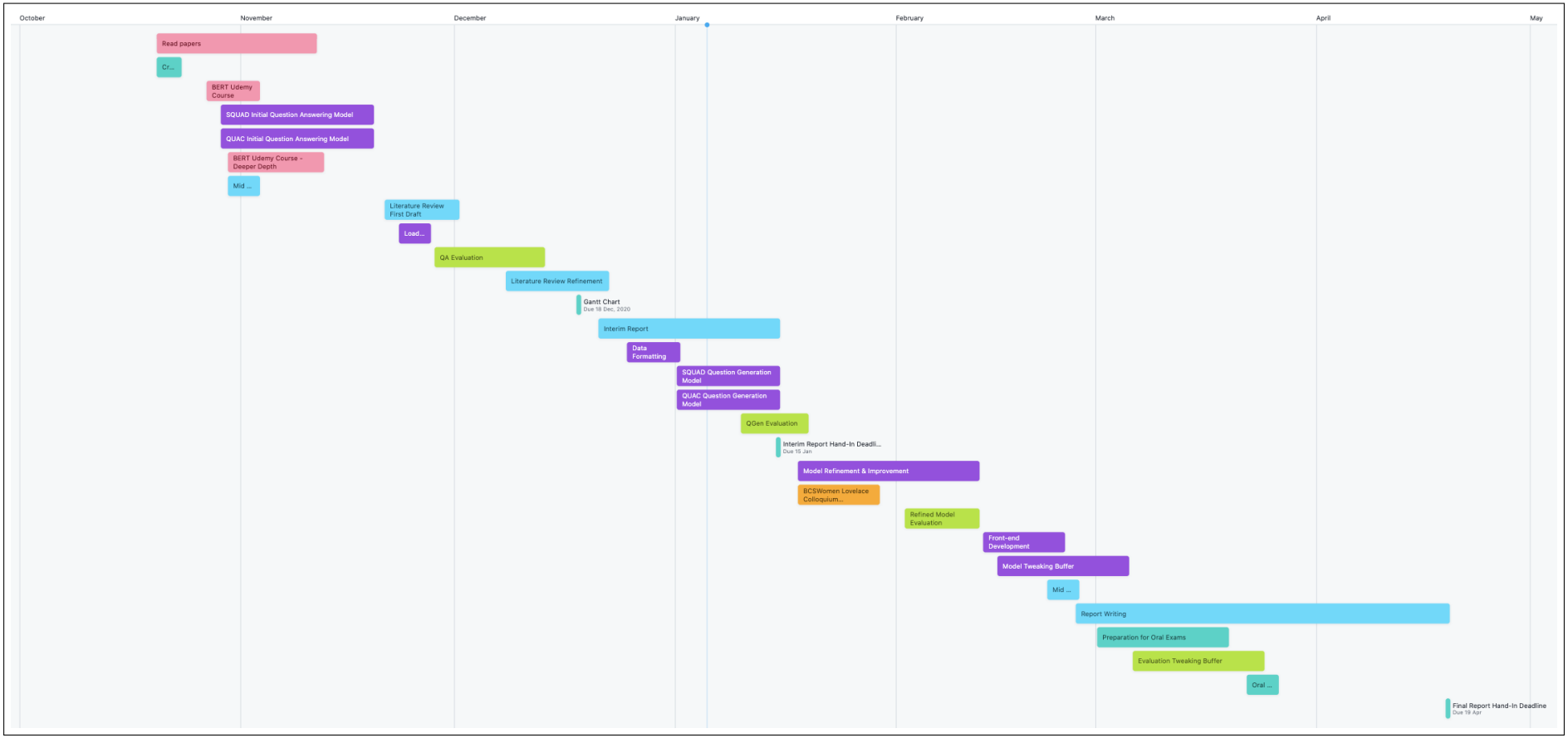

Figure 33: Project Gantt Chart

8.2 QA Reformatting Data Code

```
import json
import argparse
import sys

def parse_args():
    parser = argparse.ArgumentParser('Convert QuAC to SQUAD format')
    parser.add_argument('--input_file', help='Input QuAC JSON file', required=True)
    parser.add_argument('--output_file', help='Name of file to output', required=
                                                True)
    return parser.parse_args()

p = parse_args()

quac_file = p.input_file
out_file = p.output_file

data = json.load(open(quac_file))

paragraphs = []

for item in data["data"]:
    for paragraph in item["paragraphs"]:
        qas = []
        inner_json = {}
        for info in paragraph["qas"]:
            qas.append({"question" : info["question"], "id" : info["id"], "answers"
                                                    : [info["orig_answer"]]})
            inner_json = {"qas" : qas, "context" : paragraph["context"]}
    paragraphs.append({"title": item["title"], "paragraphs": [inner_json]})

formatted_json = {"version" : "edited_2_2_2021", "data" : paragraphs}

quac = open(out_file, "w")
json.dump(formatted_json, quac)
quac.close()
```

References

Jay Alammar. 2018. The illustrated bert, elmo, and co. (how nlp cracked transfer learning).

Jay Alammar. 2019. The illustrated gpt-2 (visualizing transformer language models).

Chris Alberti, Kenton Lee, and Michael Collins. 2019. A bert baseline for the natural questions. *arXiv preprint arXiv:1901.08634*.

Satanjeev Banerjee and Alon Lavie. 2005. Meteor: An automatic metric for mt evaluation with improved correlation with human judgments. In *Proceedings of the acl workshop on intrinsic and extrinsic evaluation measures for machine translation and/or summarization*, pages 65–72.

Steven Bird, Ewan Klein, and Edward Loper. 2009. *Natural language processing with Python: analyzing text with the natural language toolkit*. " O'Reilly Media, Inc.".

Ying-Hong Chan and Yao-Chung Fan. 2019. Bert for question generation. In *Proceedings of the 12th International Conference on Natural Language Generation*, pages 173–177.

Boxing Chen and Colin Cherry. 2014. A systematic comparison of smoothing techniques for sentence-level bleu. In *Proceedings of the Ninth Workshop on Statistical Machine Translation*, pages 362–367.

Eunsol Choi, He He, Mohit Iyyer, Mark Yatskar, Wen-tau Yih, Yejin Choi, Percy Liang, and Luke Zettlemoyer. 2018. Quac: Question answering in context. *arXiv preprint arXiv:1808.07036*.

Jacob Devlin and Ming-Wei Chang. 2018. Open sourcing bert: State-of-the-art pre-training for natural language processing.

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

AD Dongare, RR Kharde, and Amit D Kachare. 2012. Introduction to artificial neural network. *International Journal of Engineering and Innovative Technology (IJEIT)*, 2(1):189–194.

Xinya Du, Junru Shao, and Claire Cardie. 2017. Learning to ask: Neural question generation for reading comprehension. *arXiv preprint arXiv:1705.00106*.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.

Huggingface. 2021. huggingface/transformers.

Kun Jing and Jungang Xu. 2019. A survey on neural network language models. *arXiv preprint arXiv:1906.03591*.

Martin Jocqueviel, Kirill Eremenko, and Hadelin de Ponteves. 2020. Learn bert - most powerful nlp algorithm by google.

Daniel Jurafsky and James H. Martin. 2019. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Draft Book.

Tassilo Klein and Moin Nabi. 2019. Learning to answer by learning to ask: Getting the best of gpt-2 and bert worlds. *arXiv preprint arXiv:1911.02365*.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, et al. 2019. Natural questions: a benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:453–466.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2020. Albert: A lite bert for self-supervised learning of language representations.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. 2019. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *arXiv preprint arXiv:1910.13461*.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization.

Christopher Manning and Hinrich Schutze. 1999. *Foundations of statistical natural language processing*. MIT press.

Christopher D Manning, Hinrich Schütze, and Prabhakar Raghavan. 2008. *Introduction to information retrieval*. Cambridge university press.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pages 311–318.

Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*.

Chen Qu, Liu Yang, Minghui Qiu, W Bruce Croft, Yongfeng Zhang, and Mohit Iyyer. 2019. Bert with history answer embedding for conversational question answering. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and*

Development in Information Retrieval, pages 1133–1136.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

Pranav Rajpurkar, Robin Jia, and Percy Liang. 2018. Know what you don't know: Unanswerable questions for squad.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2020. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter.

Sam Schwager and John Solitario. 2019. Question and answering on squad 2.0: Bert is all you need.

Priyanka Sen and Amir Saffari. 2020. What do models learn from question answering datasets? *arXiv preprint arXiv:2004.03490*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *arXiv preprint arXiv:1706.03762*.

Tong Wang, Xingdi Yuan, and Adam Trischler. 2017. A joint model for question answering and question generation. *arXiv preprint arXiv:1706.01450*.

Svante Wold, Kim Esbensen, and Paul Geladi. 1987. Principal component analysis. *Chemometrics and intelligent laboratory systems*, 2(1-3):37–52.

Mark Yatskar. 2018. A qualitative comparison of coqa, squad 2.0 and quac. *arXiv preprint arXiv:1809.10735*.

Yang You, Igor Gitman, and Boris Ginsburg. 2017. Large batch training of convolutional networks.

Yang You, Jing Li, Sashank Reddi, Jonathan Hseu, Sanjiv Kumar, Srinadh Bhojanapalli, Xiaodan Song, James Demmel, Kurt Keutzer, and Cho-Jui Hsieh. 2020. Large batch optimization for deep learning: Training bert in 76 minutes.

Yuwen Zhang and Zhaozhuo Xu. 2019. Bert for question answering on squad 2.0.

Qingyu Zhou, Nan Yang, Furu Wei, Chuanqi Tan, Hangbo Bao, and Ming Zhou. 2017. Neural question generation from text: A preliminary study. In *National CCF Conference on Natural Language Processing and Chinese Computing*, pages 662–671. Springer.