Quickly Answering Multiple-Choice Questions using Deep Learning

Student Name: Bradley J. Mackey

Supervisor Name: Dr. Ioannis Ivrissimtzis

Submitted as part of the degree of BSc Computer Science to the

Board of Examiners in the Department of Computer Sciences, Durham University

Abstract —

Context/Background – *HQ Trivia* is a mobile application that features live daily gameshows consisting of 12 multiple-choice, general knowledge questions. Semantically complex questions—and only 10 seconds to answer each—has made this a challenge for cheaters to perfect an answering system.

Aims – This project aims to create a question answering system that is fast and accurate enough to answer questions during games of *HQ Trivia*. For this to be of practical use during live games, it must correctly determine the answer to each question and make this quickly available to the player of the game.

Method – Construction of a comprehensive question answering pipeline—based on state of the art deep neural networks and language modelling—enables open-domain questions to be answered quickly and accurately. We (1) intercept questions and possible answers during live games, using these to retrieve relevant articles from comprehensive, online corpora. We then (2) distill such articles to concise passages during "passage retrieval". Answer phrase predictions are then (3) inferred from these passages using a deep neural network language model, fine-tuned for multiple-choice "answer inference"—provided with a question and relevant contextual passage from which predictions can be informed.

Results – High quality question answering results are produced by optimising each stage of the pipeline, achieving 52% accuracy overall. This is comparable to state of the art performance in prior open-domain, multiple-choice question answering systems. Although, as no prior research specifically focuses on questions curated by *HQ Trivia*, it is difficult to establish the exact relative performance of our solution.

Conclusions – We find that our solution performs notably and conclude that the architecture is mostly sufficient given the current state of the art. Neural networks making use of language modelling perform excellently at their respective tasks in the question answering pipeline, but we find we are limited by the quality of retrieved articles; a result of the corpora used to discover relevant information. There is great scope for future experimentation and research.

Keywords — question answering, deep learning, language modelling, natural language processing, information retrieval, passage retrieval, passage re-ranking

I INTRODUCTION

The mobile gameshow application *HQ Trivia* ("*HQ*") offers large monetary rewards (up to around £100,000) to players that are able to answer all 12 general knowledge questions in any given game. The limited time to select each answer and complex semantics presented by some questions make cheating via using a search engine (including automated scripts, implementing answer-counting based approaches) a challenging task for some style of questions (Schwartz 2018). Using recent advancements in question answering research, we aim to stand the best chance at winning the jackpot[1]. As such, the research question proposed by this project is: "*Can deep neural networks be used to accurately answer questions posed by HQ Trivia?*".

A Question Answering

The domain of question answering ("QA") is one of the oldest and most researched areas in all of computer science. Early QA systems used the algorithmic analysis of questions in order to produce well-formed queries, sending these to some structured database or corpus. An early example of such a system was BASEBALL (Green et al. 1961), which deconstructed natural language questions into database queries, used to answer a select number of questions about the American baseball league at the time. However, the problem in applying this methodology to modern QA systems is two-fold:

1. the semantic complexity inherent to language makes the algorithmic analysis of questions (to create structured queries based on specific tokens) a futile task;

2. an overwhelming majority of contemporary information available today—as a result of the internet—is not structured, but in the form of natural language articles or documents.

The inherent complexity involved in language modelling and—by extension—QA has enabled the deep learning of neural networks to become the state of the art in solving such tasks. A deep neural network ("DNN") can learn representations, patterns and structural complexities within a language itself, rather than being explicitly programmed. A DNN trained for QA is able to identify a potential answer from a concise, contextual passage of text given a question.

These approaches both highlight a fundamental feature of every QA system—they depend on a database, corpus or document known to contain a given answer. For the contemporary nature of questions posed by *HQ*, the only practical way to obtain passages for a broad range of general knowledge questions is to make use of a vast and continually expanding corpus. This is known as *open-domain question answering*, as there is no single topic which questions relate to. Specific, relevant articles must be identified, on which we perform filtering to obtain a concise passage. This concise passage is then used, along with the original question, to predict a likely answer. As such, we recognise that a modern QA pipeline consists of 3 main stages: **(1) "Article Retrieval"**, retrieval of a small number of relevant articles from a large corpus; **(2) "Passage Retrieval"**, the distillation of the most relevant passages containing answers from these articles; **(3) "Answer Inference"**, precise answer identification/selection given the retrieved passages and original question.

It is worth clarifying that the last stage of this pipeline is usually just referred to as "question answering". Although, as our pipeline accomplishes the overall task of QA, we specifically refer to this stage as *answer inference* for clarity.

[1] See *Ethical Conduct*, I (D).

B Deep Neural Networks

Early efforts to apply DNNs to *natural language processing* ("NLP") based tasks used network architectures such as the convolutional neural network ("CNN") or the recurrent neural network ("RNN"). The rationale for these architecture decisions is that vanilla feed-forward neural networks (also called *multi-layer perceptrons*, an architecture from the earliest iteration of neural network research) are unable to use prior inputs to inform the network of future decisions. This is essential for sequence modelling tasks such as machine language understanding; a given word from a sentence conveys little meaning in isolation.

CNNs, RNNs and their derivates address this issue in differing ways but, in essence, both allow for network decisions to be influenced by sequences of inputs being considered as a unit rather than individually; learning abstract representations from groupings of words in a sentence. However, a key factor contributing to further research was that these network architectures struggle to learn very long-term dependencies in a given input—being unable to make associations between the start and end of a very long sentence, for example.

The advent of *attention* mechanisms allows networks to overcome this deficiency—learning relationships for inputs far separated by having inputs "attend" to others, even if not in spacial proximity. CNN and RNN based networks being augmented with these attention mechanisms help to enable clear performance increases for NLP tasks. Attention mechanisms have recently been used in architectures that are completely independent from convolutional and recurrence based network layers, a key example being the *Transformer* (Vaswani et al. 2017). These architectures place focus on training generic *language models* by unsupervised pre-training on a large quantity of language data. They can then be far more easily applied to a variety of NLP tasks via a supervised fine-tuning process.

The continued development of deep neural architectures and language models for NLP tasks—in particular QA—remains a very active area of research. Only recently have QA models been able to outperform humans in a number of key NLP benchmarks (Rajpurkar et al. 2016, Devlin et al. 2018), and we would expect this gap to widen as networks and models improve.

C Objectives

The ultimate aim of this project is to create a system that can accurately predict the answer to questions in live games of *HQ*. We place focus on the fact that these must be answered quickly (as only 10 seconds is allowed per question), although greater importance is placed on the accuracy of the final solution. This is due to increases in performance being trivial given more powerful computation hardware (such as GPU or TPU[2] acceleration); as long as the system can answer the questions within a few seconds of the time limit, time-to-answer observed during testing (on less than optimal hardware) should not be a primary concern. On the other hand, solutions that take many orders of magnitude longer than our time limit are considered unacceptable; solutions where even more powerful compute would not be sufficient to provide an answer in time.

The task specific objectives of this project closely align with the structure of the QA pipeline that we have established. These objectives were divided into three categories: basic, intermediate and advanced.

The basic objectives involved the creation of the initial QA pipeline, such that each component could then be improved independently. This entailed the creation of an article retrieval

[2] `https://cloud.google.com/tpu/`

system and algorithmic passage retriever. A language model was fine-tuned to perform *open-ended* answer inference (not considering multiple-choice answers) using a QA specific dataset; various experimentation was performed in attempt to improve the accuracy.

The intermediate objectives involved the integration with the live *HQ* question broadcasting API, such that questions could be intercepted in real-time to be input to the pipeline. In addition, a mobile application was created such that the notifications can be viewed on a client device playing the game in real-time. The passage retrieval system was also re-implemented based on a language model, fine-tuned for passage ranking using various datasets.

The advanced objectives involved the optimisation of the article retrieval and answer inference systems. Different corpora were experimented with, as well as experimentation with various querying formats. The answer inference system was adapted in order to accept the multiple-choice answers as input (in addition the question and contextual passage), to make better use of the information provided by the limited set of possible answers.

D Ethical Conduct

On the basis of ethics, legalities and morals: the system has not and will not be used to solicit any form of compensation or prize on any quiz gameshows, nor sold or provided to any party with such intent. Additionally, any eventual open-sourcing of the project will not include any *HQ Trivia* specific interoperability code. This is in accordance with sections 2.07, 2.09, 3.03, 3.04, 6.07, 6.08, 6.09, 8.07 and 8.08 of the IEEE and ACM Software Engineering Code of Ethics (Gotterbarn et al. 1997).

II RELATED WORK

Significant progress in open-domain QA has been made in recent years due to research advancements in the field of deep learning for NLP, as well as vastly improved datasets for QA and passage retrieval based tasks. Lessons learned from prior attempts to beat *HQ*, as well as these state of the art research advancements in QA and passage retrieval, inform the architecture of our solution.

A HQ Trivia QA

As *HQ* has been prevalent in popular culture since its launch in 2017, attempts have been made by some individuals to create automated answering systems. To date, this mainly consists of using simple scripts to perform repeated search engine querying (Schwartz 2018). This naive approach is shown to be effective for some questions, where a greater number of results relating to a certain topic unambiguously implies correctness for a given multiple-choice answer. Obviously, more semantically complex questions are a challenge for such a system. A functional contribution offered by Schwartz (2018) are details relating to the *HQ* gameshow API and their proprietary message structures; this allows for questions to be directly read from a WebSocket during live games without the need for time-consuming, OCR-based screen reading.

B Language Modelling

As discussed, DNNs currently offer the best performance in NLP tasks—a result of the complexities and nuances involved in language modelling. Before the advent of *Transformer* based

networks (Vaswani et al. 2017, Radford et al. 2018, Devlin et al. 2018), the RNN derived long short-term memory ("LSTM") network (Hochreiter & Schmidhuber 1997) was considered state of the art for NLP; the highest performing approaches worked towards the creation of generic language models. LSTM based networks were shown to be effective at large scale language modelling (Józefowicz et al. 2016), ELMo (Embeddings from Language Models) (Peters et al. 2018) being an example of an LSTM based language modelling architecture. ELMo architectures consist of many bidirectionally trained LSTMs in order to deeply model language structure.

More recently published transformer based models, such as the Generative Pre-Trained Transformer ("OpenAI GPT") (Radford et al. 2018), have noticeable performance improvements over LSTM based models. The current best performing network is BERT (Bidirectional Encoding Representations from Transformers) (Devlin et al. 2018), featuring deep bidirectionally connected Transformers; inspired both by ELMo and the OpenAI GPT, it enables more informative attention-based connections and provides state of the art NLP performance. As of early 2019, BERT currently forms the basis of a number of models achieving recording-breaking results in a number of NLP tasks. Notable tests showcasing this include: the General Language Understanding Evaluation ("GLUE") (Wang et al. 2018) NLP challenges, Microsoft Machine Reading Comprehension Dataset ("MS MARCO") passage re-ranking (Nguyen et al. 2016) and Stanford Question Answering Dataset ("SQuAD") (Rajpurkar et al. 2016) benchmarks. BERT is a pre-trained model architecture, meaning it is quickly adapted to different NLP tasks via fine-tuning.

C Corpora/Knowledge Bases

Many open-domain QA systems make use of Wikipedia as their sole source of articles from which to extract passages to subsequently answer questions (Yang et al. 2019, Chen et al. 2017). Wikipedia being such a comprehensive source of knowledge, containing over 5.8M articles (2019)[3], makes this an appropriate choice for these pipelines—able to achieve impressive results in similar QA systems. Chen et al. (2017) implement an article retriever that uses an archived, indexed dump of Wikipedia which is used to identify articles via an inverted index lookup, followed by term vector scoring. The authors note that this performs slightly better than the default ElasticSearch[4] based Wikipedia API for finding articles, indicating the built-in API's search feature is less than optimal.

The pipelines using Wikipedia use this due to a number of factors, including: the broad range of article topics available, ease of accessibility of Wikipedia articles via their public API, depth of knowledge in articles and the fact that many QA datasets, including SQuAD, have questions sourced from Wikipedia (Yang et al. 2019, Chen et al. 2017, Rajpurkar et al. 2016). Although, restricting potential articles to a single knowledge base negatively impacts performance if the answer to a given question cannot be found.

D Passage Retrieval/Ranking

A passage retrieval system is an integral aspect of any open-domain QA system, distilling retrieved articles to only a few relevant sentences; these passages are then used to answer the question during "answer inference". Filtered passages must be concise enough for the answer

[3] `https://en.wikipedia.org/wiki/Wikipedia:Statistics`
[4] `https://www.elastic.co/products/elasticsearch`

inference stage to quickly retrieve a likely answer. There are two main approaches for passage retrieval that are applicable to our system: algorithmic and DNN based.

Algorithmic approaches are widely used in this field due to their speed and retrieval accuracy, a notable and established example being BM25 (Robertson & Walker 1994). The original design of BM25 was intended to rank web documents, computing a relevance score by taking into account the term frequency (TF)—the frequency of relevant query terms—and inverse document frequency (IDF)—the rarity of relevant query terms—within a given document. For this reason, BM25 is known as a TF-IDF retrieval function—a common approach used in text mining applications (Mitra & Craswell 2017). BM25 is also considered an *exact matching* algorithm, as documents are scored directly based on the specific keyword search terms that appear in them; weaknesses include synonyms of possible query words being ignored and out of context words being considered matches. Mitra & Craswell (2017) highlight that these exact matching models are very effective at retrieving passages containing rare terms (such as a person's surname or unusual other term, common occurrences in a trivia game such as *HQ*). Additionally, Yang et al. (2019) conclude that filtering retrieved articles via paragraph, not sentence, yields increased accuracy during retrieval.

BM25F is a variant of BM25, applying differential weighting to keywords based on where that word appears in the document (for example, titles are considered more important than the body of the text) (Mitra & Craswell 2017). This is able to garner improved performance over non-weighted TF-IDF ranking models when ranking articles from Wikipedia.

In contrast, DNN approaches based on language models are able to provide vastly improved ranking accuracy at the expense of increased computation time. Nogueira & Cho (2019) demonstrate this by fine-tuning a BERT model for passage re-ranking using the MS MARCO dataset (Nguyen et al. 2016) which includes: questions, passages and scores for how relevant a given passage is for each question. They observe state of the art performance, improving over the prior best model by a significant margin (and vastly out-performing BM25 based approaches). They concluded that only a fraction (0.3%) of the entire training dataset was required for fine-tuning to out-perform the previous state of the art, as MS MARCO contains a vast number of training examples.

E Answer Inference

QA largely being accomplished via DNN based approaches has necessitated the need for large and complex labelled datasets to enable high accuracy in a variety of domains; training data is a key defining property of a given model's performance (Goodfellow et al. 2016). The increase in the size and improvement in quality of open-source QA datasets is undoubtedly a factor in the improved accuracy and research breakthroughs in recent models (Devlin et al. 2018).

As mentioned, the SQuAD benchmark (Rajpurkar et al. 2016) for QA is widely referenced by researchers in the field due to its large size (100K examples) and comprehensive questions; it has been used to train state of the art models for QA: a single BERT model is able to achieve test accuracy of 85% when fine-tuned on SQuAD 1.1 (Devlin et al. 2018). Larger datasets have since been released with more semantically complex questions. TriviaQA (Joshi et al. 2017) is such an example, containing 650K questions. Indeed, BERT achieves highest accuracy on the SQuAD 1.1 benchmark when trained both on SQuAD and TriviaQA, shown in the original BERT paper. These datasets train using span-based answers, where a possible answer to the question is located directly from the text itself.

Recent research into applying BERT specifically for multiple-choice QA has had fruitful results. The RACE dataset (Lai et al. 2017) consists of 100K questions in a similar style to SQuAD, differing by including possible answers for each. As of February 2019, the best performing model when tested on RACE is, unsurprisingly, BERT based; achieving accuracy of 72%—showing RACE is undoubtably a more difficult QA task than SQuAD. This model (Pan et al. 2019) was initially fine-tuned on multiple-choice questions from RACE, then adopted a *transfer learning* technique, fine-tuning on two further multiple-choice datasets independently in order to perform well on both of these benchmarks. Transfer learning follows a similar philosophy to that of fine-tuning: an existing trained model is trained again on a task specific dataset, but maintains the weightings of the existing model, learning enough additional information to perform the task defined by the new dataset instead. This eliminates the training requirement for the entire network, meaning this is much faster than standard training. The datasets which transfer learning was performed on are ARC and OpenBookQA. Notably, ARC does not include supporting passages for each question. Although, a similar dataset—SciQ—contains questions of a similar style to that of OpenBookQA and does include supporting passages. This makes SciQ a far better candidate for any multiple-choice answer inference training.

Ultimately, the best performing open-domain QA solutions are highly complex, usually consisting of ensembles of different systems at each stage of the pipeline to avoid pitfalls present in any given technique (Chen et al. 2017, Devlin et al. 2018).

III SOLUTION

In order to create a system capable of quickly and accurately answering live questions from *HQ*, we construct a pipeline similar to that which we have outlined. The first stage intercepts the live gameshow questions as quickly as possible from *HQ*, bearing in mind the 10 second question time limit. The second stage is article retrieval; we use the question and possible answers to query a corpus for a small number of relevant articles likely to contain the answer. The third stage is passage retrieval; we use a combination of algorithmic and DNN based approaches (optimising for both retrieval accuracy and speed) to filter the most relevant passages. The penultimate stage feeds the retrieved passages, the original question and multiple-choice answers to a DNN trained for QA to produce a likely prediction as to the best possible answer. Finally, the player of the game is notified of this prediction via a push notification directly to their gameplay device so they can, theoretically, use this to improve their odds of winning. An overview of the architecture can be viewed in Figure 1.

Implementation was not performed in the linear fashion outlined here; we first produced a minimum viable product involving all stages, then gradually made improvements to each independently. This allowed for us to monitor overall performance changes to the system during experimentations.

A Tools and Technologies

The primary technology involved in the development of this system is the programming language, coordinating all aspects of the pipeline. This language was Python, for which there were a multitude of reasons; the main being that the most well known and best maintained deep learning libraries include support for/are only available for Python. Several open-source deep learn-

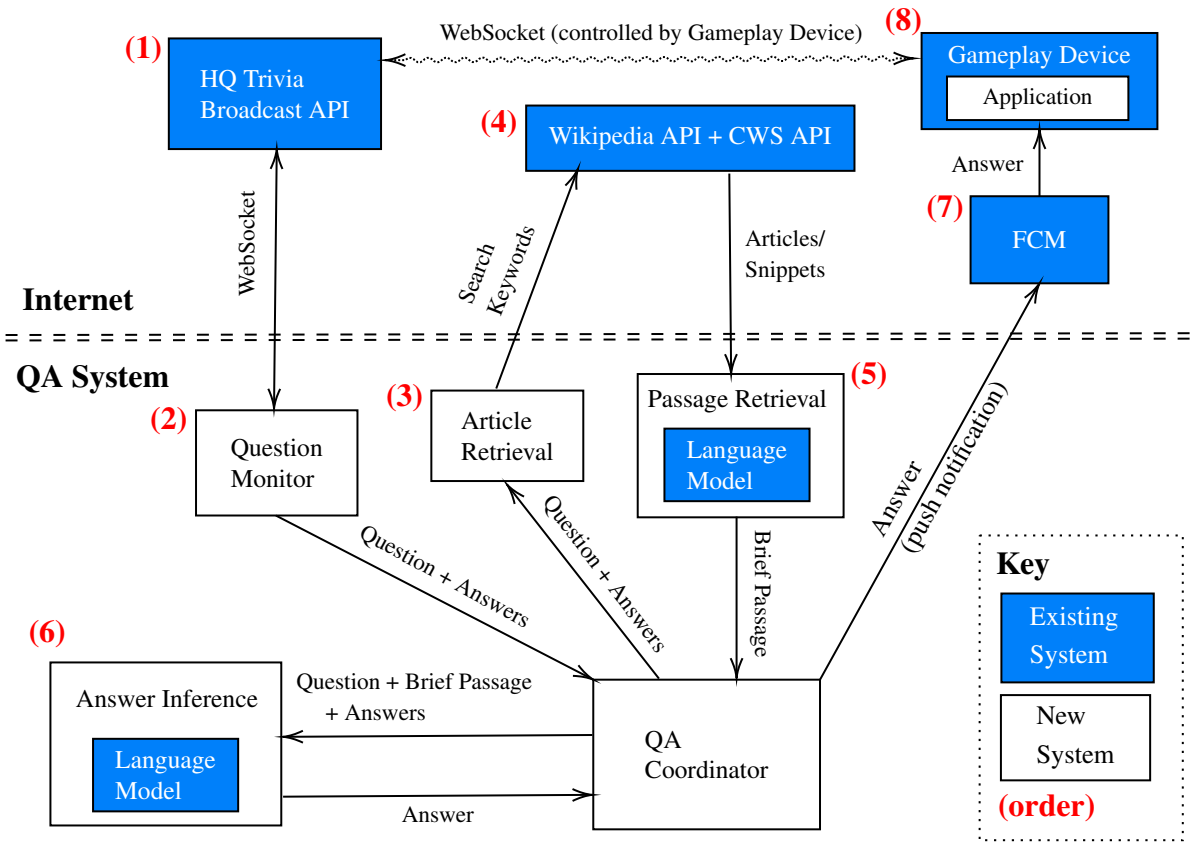

Figure 1: System architecture pipeline, highlighting main components and interactions between these. The data flow as a question is intercepted and answered is also highlighted (the connection between the Gameplay Device and the *HQ Trivia* Broadcast API is not managed by us, and is only established so the user can play the game in real-time as predictions are generated). Articles, from which ground truths are established, are obtained via the Wikipedia and Contextual Web Search (CWS) public APIs.

ing libraries were available to facilitate the creation of the networks and models required for this project. The framework used for this project was PyTorch (Paszke et al. 2016), opposed to popular competing framework TensorFlow. This is due to the beginner friendly API, wealth of documentation available and community support—particularly among researchers. Additionally, some base model architectures required for this project—such as BERT (Devlin et al. 2018)—have been created, open-sourced and supported by the PyTorch community. To allow for seamless compatibility and to reduce latency, the remainder of our system was also implemented using Python. The tokenisation of words during the article retrieval phase was performed using the `nltk` library, allowing for fast and accurate NLP tasks to be performed on text.

In order for a client device to receive answer predictions during live games, an application was created as a stub for push notifications to be sent to. The platform we target with this stub application is iOS, written in the Swift language. This decision was due to us having an iOS device readily for testing and prior experience in creating applications for iOS; stub applications could trivially be written for other platforms to receive the push notifications. Push notifications sent from the QA system are forwarded though the Firebase Cloud Messaging ("FCM") service as a proxy before being delivered to the device, rather than having to directly interface with

platform specific notification services (which, in the case of an iOS device, is the Apple Push Notification Service). FCM is a free, cross-platform push notification service, and easily allows for new platforms to be supported if client specific applications are written for them. When testing the latency of FCM (to see if it is suitable for fast answer communication) we found this was consistently below 100ms, which is fast enough for our system.

B Article Retrieval

To answer a given open-domain question, we must obtain a relevant, contextual passage containing the answer from an authoritative corpus. Given that, in the case of *HQ*, this can relate to any facet of any particular topic, this corpus must be vast and have suitable depth of knowledge.

As demonstrated in end-to-end QA papers (Chen et al. 2017, Yang et al. 2019), Wikipedia is an excellent candidate for this knowledge source. However, having access to the very latest article edits would require using Wikipedia's standard API to query articles; persistently updating and indexing a local dump of the entire corpus is not practical for this project, unlike the approach by Chen et al. (2017)—despite this approach presenting improved performance.

Using Wikipedia as a single corpus is unnecessarily restrictive for the class of questions presented in *HQ*, which can relate to highly contemporary topics (for which Wikipedia may not yet have been updated). Expanding to use multiple corpora in such a pipeline will enable more questions to be answered. Contextual Web Search[5] is an inexpensive search engine API, allowing for indexed documents and articles to be returned. Using the top few search results from a query derived from keywords in the question and potential answers, we download the content of these pages concurrently and parse the HTML to generate further passages to draw from. This provides us with a greater breadth of up-to-date information than Wikipedia alone, which proved valuable in increasing answering accuracy.

In order to effectively retrieve articles from these sources, query terms must be created from the information we have intercepted from *HQ*—the question and potential answers. To achieve this, we use the Python `nltk` library to perform tokenisation of the question. We elected not to use a language model for query creation due to this task not being well defined (a "good" structure for a query will vary depending on the topic and availability of information), lack of any possible training datasets and algorithmic approaches providing good results—thanks to the powerful searching abilities already provided by our corpora. We extract nouns and named entities from the question; using these—along with the multiple-choice answers—we query our corpora directly, not specifying any specific ordering of these terms (found to make little difference). We then remove words of length less than 3 from the search, which were found to be one of the main causes of unrelated articles. Negating words (such as "not", "isn't") are also removed from queries, which improves performance as the semantics of negation are deferred until the answer inference stage; its sophisticated language model determines how the negation correctly relates to the subject of question. The *stemming* of search terms (reducing each word to its root) was also explored, but this was found to have a negative impact on overall performance.

C Deep Language Model

Due to demonstrating state of the art performance in both passage re-ranking and answer inference tasks, we use **BERT** as our core language model (Nogueira & Cho 2019, Pan et al. 2019, De-

[5] `https://contextualwebsearch.com`

vlin et al. 2018).

BERT follows a pre-training approach for language modelling, where the model must be initially trained in an *unsupervised* manner on an extremely large quantity of language data, then fine-tuned to a specific NLP task. Unsupervised learning does not include any human-labelled datasets; a model is given some objectives on some unlabelled dataset in order to learn the structure of such data. In the case of BERT, this involved (1) next sentence and (2) masked word predictions—these alone allow for the comprehensive modelling of a given language. The pre-training datasets selected by the authors were the entirety of Wikipedia and BooksCorpus, totalling over 3.2B words—this procedure is expensive, taking many days to train on powerful TPU hardware. The authors have publicly released pre-trained models, allowing their use without this overhead. These pre-trained models are $\text{BERT}_{\text{BASE}}$ and $\text{BERT}_{\text{LARGE}}$, differing in terms of their internal structure. $\text{BERT}_{\text{BASE}}$ features 12 Transformer layers and 110M network parameters (used for language modelling), while $\text{BERT}_{\text{LARGE}}$ features 24 Transformer layers and 340M network parameters. This results in $\text{BERT}_{\text{LARGE}}$ being able to achieve greater performance by having a greater capacity for language modelling, while $\text{BERT}_{\text{BASE}}$ is faster to train. Unfortunately, as we were constrained to (at best) a single NVIDIA Tesla V100 (16GB) GPGPU during the project, fine-tuning was only possible using the $\text{BERT}_{\text{BASE}}$ pre-trained model.

The model is relatively quickly fined-tuned to specific NLP tasks by training the model in a *supervised* manner using a task-specific, human-labelled dataset. Fine-tuning only slightly adjusts the internal weights of the model in order to produce predictions based on that dataset; the language modelling properties of the network are maintained, albeit with slightly modified values. As a result, experimentation with deep neural networks for a variety of NLP tasks can be performed in an increasingly expeditious and economical manner than prior methods.

When fine-tuning BERT for a given prediction task, the model expects the input to be in a particular format depending on the task, tokenised using WordPiece *embeddings*. An embedding is a representation for textual information that is input to a neural network. As neural networks only operate on *tensors* (any-dimensional arrays of numbers), WordPiece maps word tokens to numerical identifiers determined by the trained network, such that the values correspond. Semantic meaning and relations between these embedded numerical values are encapsulated by the pre-trained model, rather than being self-contained in the embedding (unlike other approaches, where embeddings are many-dimensional vectors). Figure 2 shows this fine-tuning procedure on SQuAD 1.1, inputting embedded tokens and outputting a range of the passage believed to be the answer.

D Training Datasets

When training or fine-tuning DNNs based on language models, high quality labelled datasets are required in order to obtain high quality results—the performance of a network is directly dependent on the quality of the training data.

Depending on the task at hand, differing datasets are needed; separate parts of the pipeline fundamentally have different tasks and, as such, need to be trained on different data. These are discussed in the detail parts of the pipeline that these datasets relate to. As we do not pre-train the BERT model ourselves, there is no requirement for us to obtain the data for this task.

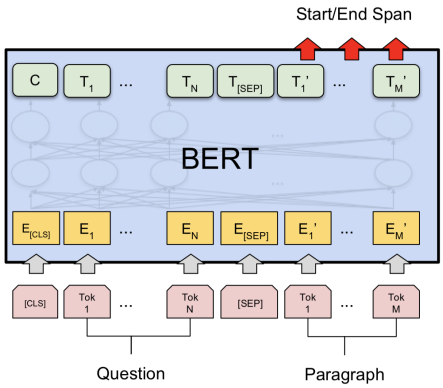

Figure 2: BERT fine-tuning on SQuAD 1.1 QA dataset (Devlin et al. 2018). The tokenised question and passage of text are input to the model; the model produces a predicted span of text from the passage as an output. As BERT can be fine-tuned for a variety of NLP tasks, the format of this input varies depending on the fine-tuning task at hand. Notably, SQuAD 1.1 is not a multiple-choice dataset and, as such, no potential answers are input to the network in this example.

E Neural Network Training

Training of DNNs is a very computationally intensive process but can easily be parallelised to run on a general-purpose GPU ("GPGPU"), which contain many cores to perform operations in parallel. The PyTorch framework contains first-class support for GPGPU training. In order to train/fine-tune our models we make use of Durham University's NVIDIA CUDA Centre ("NCC"), offering powerful GPGPUs for use on a time-share basis. This proved extremely helpful in reducing the amount of time required for model training, which remained the main bottleneck during experimentations. When GPGPUs on NCC were busy or to more quickly debug training code, we made use of the cloud infrastructure provider AWS (billed by usage).

The training process itself adjusts the internal weights of each model such that network predictions minimise a given *loss function*. The loss of a network is calculated by comparing the output to ground truth, defined for each example in a given training dataset. The ultimate aim of the training is to create a model that generalises well to all inputs of a given class—not just examples similar to that of the training dataset. This is the primary reason for requiring a large, diverse dataset for any given task; helping to ensure generalisation.

Training was also monitored so we could verify that *overfitting* did not occur. Overfitting takes place when a model is trained for too many epochs (iterations) on any given dataset; the network's performance on prior unseen examples degrades, it merely performs better specifically on examples from the training set. Overfitting was prevented by training models for varying numbers of epochs and inspecting relative performance changes between these epochs.

F Passage Retrieval

Before a question can be answered using the retrieved articles via a fine-tuned DNN, we filter these articles, such that only a concise and relevant paragraph is used for answer inference. While it is theoretically possible each entire article could be input along with a given question during answer inference, we found during testing this results in a far greater number of false positives

and takes an unacceptably long time. Therefore, we perform passage ranking in order to return the most relevant paragraphs (rather than sentences; for improved accuracy) (Yang et al. 2019).

The BM25F ranking algorithm is a fast, established and accurate document relevance scoring system, typically used within search engines and full-text querying systems. For a given query Q containing keywords $\{q_1, ..., q_n\}$, the score of a paragraph P is:

$$\text{BM25F}(P, Q) = \sum_{i=1}^{n} \text{IDF}(q_i) \cdot \frac{tf(q_i, P) \cdot (k_1 + 1)}{tf(q_i, P) + k_1 \cdot \left(1 - b + b \cdot \frac{|P|}{\text{avgpl}}\right)} \tag{1}$$

The value $tf(q_i, P)$ is q_i's weighted term frequency in paragraph P, $|P|$ is the length of paragraph P in words and avgpl is the average paragraph length from all retrieved articles. The variables k_1 and b are free parameters, we set $k_1 = 1.5$ and $b = 0.75$ (experimentally determined). IDF(q_i) is the inverse 'document' (paragraph) frequency of a query term q_i:

$$\text{IDF}(q_i) = \log \frac{N - n(q_i) + 0.5}{n(q_i) + 0.5} \tag{2}$$

where N is the total number of paragraphs and $n(q_i)$ is the number of paragraphs containing q_i. The term frequency $tf(q_i, P)$ is weighted based on 'importance' values depending on where a query word appears, from the set of fields $F = \{\text{title}, \text{body}, \text{footnote}, ...\}$:

$$tf(q, p) = \sum_{f \in F} W_f \cdot x(q_{p,f}) \tag{3}$$

where $q_{p,f}$ is a query term appearing in paragraph p within field f and W_f is the weighting of field f and x is a function that counts the frequency of this term. Term frequency can also be defined by a number of other metrics (Mitra & Craswell 2017), one such being the *boolean frequency*; if the count of $q_{p,f}$ is greater than 0, x only returns 1. We verify this ranking algorithm by analysing performance for a number of Wikipedia articles relating to ***HQ*** questions, success is determined if any of the returned passages entail the answer to the question. BM25F proved to be an effective algorithm at ranking passages to allow the answer inference stage to pick the best possible answers.

We also experimented with filtering such results using DNNs in order to improve performance. Firstly, filtering the 12 most relevant paragraphs, found to be an optimal number, using BERT fine-tuned on the MS MARCO passage re-ranking dataset (Nguyen et al. 2016, Nogueira & Cho 2019). A fine-tuned BERT model is currently the best-performing re-ranker on the MS MARCO dataset, producing a probability p_k of a passage k being relevant. We also experimented with a BERT model fine-tuned on the GLUE (Wang et al. 2018) Question Natural Language Inference ("QNLI") NLP task. This is a task that constitutes classifying whether a given passage entails an answer, which can also be used to filter potential candidate answers. The dataset for QNLI is derived from that of SQuAD, using the same questions and passages of text. DNN based rankers showed improved performance in isolation, but gave unfortunately disappointing results when used in tandem with BM25F. The timing requirements of the system prevented us from using a DNN based ranker for all passages, due to the vast amount of computation required.

G Open-Ended Answer Inference

The initial answer inference DNN was trained for open-ended QA, based on BERT and fine-tuned on the SQuAD and TriviaQA datasets. This is due to the fact that excellent QA performance had

been observed using these datasets (Devlin et al. 2018) and these datasets being more extensive and established than those available for multiple-choice QA. This would also serve as a good benchmark compared to multiple-choice answer inference, specifically in the domain of general knowledge *HQ* questions. As SQuAD and TriviaQA have their training examples stored in differing and proprietary formats, we first converted these to a common structure. This is such that examples could be efficiently read from both training sets in a single batch and fine-tune the network all at once.

As made clear in Figure 2, the output of this trained network are spans of text from the original passage and the estimated probability of these being correct, according to the dataset the network was fine-tuned on. In order to ensure that this output is relevant given the multiple-choice answers to the question, we performed *fuzzy text matching* on all predicted answers to the possible answers. Fuzzy text matching gives a similarity score for two given strings, which is helpful for matching mostly similar answers, even if the phrasing is different than required (e.g. "**Fr**e**nc**h" vs. "**Fr**a**nc**e"). The fuzzy matching algorithm we found to be the most accurate was 'partial token set ratio'. This ignores word order in each phrase and does not consider the entire phrase for each match—as we found that predictions, on occasion, can be long phrases that briefly feature the correct answer. This ensures that the best possible prediction is matched to the closest one of our possible answers.

H Multiple-Choice Answer Inference

We also implemented multiple-choice answer inference in order to use the information provided provided by these possible answers, as stated in our advanced objectives. We use this as a basis for comparison against open-ended answer inference to determine which model architecture performs better for *HQ* questions. Similar to the approach described by Pan et al. (2019), we first fine-tune BERT using the RACE dataset, then perform subsequent transfer learning on OpenBookQA and SciQ—all multiple-choice QA datasets with supporting passages. While the style of questions presented in the OpenBookQA and SciQ datasets is much closer to that seen in *HQ* (in comparison to RACE, a multiple-choice sentence completion task), these datasets are much smaller: containing only 6K and 14K questions respectively, while RACE contains over 100K examples. This makes the transfer learning approach a sensible choice for our application—the more complex and comprehensive the training dataset, the better a given model is able to perform when answering unseen questions (Goodfellow et al. 2016, Joshi et al. 2017).

The output of this network differs from open-ended answer inference; a probability vector is generated, giving the likelihood of each multiple-choice answer being correct. This is a more robust approach, as the network knows to only consider the available answers when creating predictions. With the output being given in terms of probabilities, we are able to test the correctness of this network by ensuring that the length of this vector is always 3 (the number of multiple-choice answers per question in *HQ*) and that these probabilities always sum to 1. This model is ultimately evaluated on questions sampled directly from live games of *HQ*.

IV RESULTS

The pipelined nature of our solution compels us to evaluate each stage of the pipeline independently to identify bottlenecks, then to assess the overall performance of the system to determine suitability to the problem domain. As all stages of the QA system rely on producing a result

relative to some oracle, we use the exact match (EM) score to measure the pure accuracy of our system. This is simply defined as the number of answers the system correctly produces compared to the total number of predictions:

$$\text{EM} = \frac{\text{correct}}{\text{correct} + \text{incorrect}} \tag{4}$$

It is commonplace that open-ended QA systems are benchmarked against their "F1" score, a measure of both the precision and recall in a system. However, as F1 is a *binary classification* metric, it is not defined for our multiple-choice QA task where an answer is **always** produced; false positives or negatives are not well defined. Therefore, to provide a basis of comparison between our open-ended and multiple-choice approaches, we only consider EM scores.

The testing of each stage would ideally only use questions sampled from *HQ* directly. However, we do not have access to a comprehensive dataset of accurately retrieved passages for questions of *HQ*; we only hold a moderately sized dataset (1000 examples) of questions, multiple-choice answers and the correct answer. Therefore, we test using the datasets from the domains which each stage of the pipeline focuses on (such as the test dataset for the task that a particular DNN was trained on); we also show how this affects the overall answering performance of our system using our *HQ* questions. The test data for pre-existing datasets were verified to have roughly a 70-30 test-train data split (a standard split ratio for machine learning) to ensure that networks are optimally trained and and test results generalise well to real-world performance. Testing at all stages was performed on a 2018 6-core 2.2GHz i7 MacBook Pro, without CUDA acceleration and turbo-boost disabled (far from optimal hardware); time reduction at various stages using more powerful compute hardware is trivial. The average time for a question to be intercepted by WebSocket from *HQ* was 150ms and the time for a notification to be delivered and visible on a user's device was 500ms, fast enough for our application.

A Article Retrieval

The evaluation and optimisation of the article retrieval system was the last task performed, such that changes in overall performance for *HQ* questions could be used as a benchmark. This is due to the dependency on the passage ranking and answer inference stages, which ultimately determine the answer from any number of possible passages. Each experiment was performed over a random sample of 250 of the 1000 *HQ* questions, to efficiently utilise time.

Table 1 clearly shows the combination of both Wikipedia and CWS allowing for improved search results overall, with both providing a wider base of knowledge than compared to a single corpus. Named entities and nouns were found to be the minimum amount of information required for remotely relevant articles to be returned at all, due to them so often containing the subject of a given question. A clear finding is that the inclusion of verbs (extracted from the question) in the search query degraded the performance of articles from both corpora; nouns and named entities were sufficient to find relevant articles. We also note that increasing the number of retrieved articles via CWS degrades answering performance if made too high, due to polluting the passage ranker with more false positive passages. The degree to which stemming reduced performance is clearly visible as well, reducing accuracy and dramatically increasing the search time. This highlights the degree to which article quality affects the passage ranking and subsequently overall answering performance.

Table 1: Corpora performance when varying parameters. Query words are: nouns (N), named entities (NE), verbs (V) and whether these query words were stemmed (S). # Articles is the number of articles retrieved from each corpus, where `#QW` is the number of query words used in the search, derived from the question and possible answers, of which we retrieve a Wikipedia article for each.

Corpora Used	Query Words	# Articles	Avg. Time	EM
Wikipedia	N+NE	`#QW`	1.2s	0.40
Wikipedia	N+NE (S)	`#QW`	26.3s	0.28
Wikipedia	N+NE+V	`#QW`	1.2s	0.37
CWS	N+NE	5	2.4s	0.40
CWS	N+NE	11	4.6s	0.46
CWS	N+NE (S)	11	32.3s	0.44
CWS	N+NE+V	11	4.7s	0.45
CWS	N+NE	16	6.9s	0.42
Wikipedia + CWS	N+NE	`#QW` + 11	5.8s	**0.52**

B Passage Retrieval

We evaluate the passage ranking techniques by testing against the evaluation set of the MS MARCO passage re-ranking benchmark. Passage ranking is typically measured using the mean reciprocal rank ("MRR") metric, able to score the correctness order of a given list effectively for a sample of queries Q:

$$\text{MRR} = \frac{1}{|Q|}\sum_{i=1}^{|Q|}\frac{1}{\text{rank}_i} \tag{5}$$

where rank_i is the rank position of the first relevant document for the ith query.

Table 2: Passage ranking results on the MS MARCO passage re-ranking benchmark. The term frequency counting method for BM25F is specified, the DNNMSM (DNN MS MARCO) is the pre-fine-tuned $\text{BERT}_{\text{LARGE}}$ model for MS MARCO ranking.

Method	Term Freq.	Fuzzy Match	MRR
BM25F	count	✗	0.186
BM25F	count	✓	0.189
BM25F	boolean	✗	0.189
BM25F	boolean	✓	0.192
DNNMSM	-	-	**0.356**

Table 2 clearly shows that the $\text{BERT}_{\text{LARGE}}$ model fine-tuned for MS MARCO ranking (Nogueira & Cho 2019) vastly outperforms the performance of BM25F, no matter the term-frequency function. Indeed, this was the research aim of Nogueira & Cho (2019). We also see that when BM25F is allowed to near match terms via fuzzy matching, slight performance gains are able to experienced. Specifically, we require a 90% partial match of a query word within a passage for a the term to count (experimentally determined to be optimal). The GLUE QNLI passage entailment model is not included in this benchmark as this produces a simple binary output, declaring if a given passage entails some question; it is not used for passage ranking.

To test which combination of ranking algorithms performed optimally, we experimented on the extracted *HQ* questions test set only when the other stages of the pipeline already optimised. The computationally intensive DNNMSM and QNLI systems were used as post-filtering stages from a BM25F pre-fetch.

Table 3: Overall QA accuracy for pipeline based on ranking algorithm, tested on *HQ* test questions. DNNMSM is defined previously and QNLI is a $\text{BERT}_{\text{BASE}}$ model trained on GLUE QNLI. The number for each stage is the maximum number of paragraphs returned by each stage.

Ranker #1	**Ranker #2**	**Avg. Time**	**EM**
BM25F (6)	-	0.45s	0.43
BM25F (12)	-	0.45s	**0.52**
BM25F (18)	-	0.45s	0.48
BM25F (25)	DNNMSM (6)	20.13s	0.44
BM25F (25)	DNNMSM (12)	20.14s	0.46
BM25F (25)	DNNMSM (18)	20.32s	0.45
BM25F (25)	QNLI (6)	14.06s	0.42
BM25F (25)	QNLI (12)	14.07s	0.47
BM25F (25)	QNLI (18)	14.06s	0.45

The results of Table 3 are somewhat unintuitive. They clearly show that adding a post-filtering stage after BM25F reduced performance for QA overall. As BM25F is first used to retrieve a small number of relevant passages, the additional ranking stages depend on this initial fetch being accurate; this may not always be the case, which hinders performance for later stages. The DNNMSM ranker was not used in isolation due to taking an impractical amount of time to rank all passages during testing; on average, several thousand are usually returned during article retrieval. The number of passages retrieved during this stage was found to be optimal at 12; a great number of experiments were performed to optimise this retrieved number; we highlight some key results here. A clear takeaway is that if too many paragraphs are returned, answer inference is less effective due to an increased number of distracting or misleading passages being present. We found this optimal BM25F ranking and filtering system took on average less than 500ms.

C Answer Inference

The two main approaches for answer inference—outlined in our solution—involve using a fine-tuned $\text{BERT}_{\text{BASE}}$ model trained for both open-ended and multiple-choice answering tasks. We first aimed to determine which combinations of fine-tuning datasets were the most effective for each specific paradigm, in order to give the greatest chance at success for questions from *HQ*. As open-ended and multiple-choice answer inference are fundamentally different tasks, we use differing datasets to benchmark each independently.

Table 4 gives clear evidence that the greater number of relevant datasets a given BERT model is fine-tuned on, the higher accuracy will be achieved during QA. Benchmarks for both of these paradigms were performed on the SQuAD and SciQ benchmarks, both similar in nature to questions that are asked during games of *HQ*. However, the results between these two cannot be

Table 4: QA performance for different model paradigms and datasets for answer inference.

Paradigm	Fine-Tuning Datasets	Benchmark	EM
Open-Ended	SQuAD	SQuAD	0.80
Open-Ended	SQuAD+TriviaQA	SQuAD	0.84
Multiple-Choice	RACE+SciQ	SciQ	0.91
Multiple-Choice	RACE+OpenBookQA+SciQ	SciQ	0.93

compared, as the test questions fundamentally remain different; relative gains can be seen however.

In order to compare these different paradigms in terms of overall answering performance, testing was performed on the entirety of the *HQ* test dataset, with the other aspects of the pipeline already fully optimised.

Table 5: Overall QA performance on the *HQ* test set, from the best model for each QA paradigm.

Paradigm	Fine-Tuning Datasets	Avg. Time	EM
Open-Ended	SQuAD+TriviaQA	4.03s	0.46
Multiple-Choice	RACE+OpenBookQA+SciQ	2.53s	**0.52**

Table 5 clearly outlines that our multiple-choice question answering system is the most optimal for questions from *HQ*. We find that performance is, again, mainly limited by retrieved passage quality. Indeed, the multiple choice network is able to obtain 93% accuracy on the SciQ multiple-choice benchmark test set (Table 4).

Our open-ended networks used fuzzy matching to consolidate predicted spans with the multiple-choice options for each question, with a required similarity threshold of 70%. This was experimentally determined to be optimal as answers are often contained in a longer string or answers may not be an exact match.

V EVALUATION

Here, we shall evaluate the strengths and weaknesses of our solution to determine if we have been able to effectively answer our original research question: "*Can deep neural networks be used to accurately answer questions posed by HQ Trivia?*".

A System Strengths

We have demonstrated a QA pipeline that is able to answer a majority of the questions posed by *HQ* correctly. We have independently verified recent research in the field of NLP that language modelling based DNNs—BERT in particular—perform excellently at their respective tasks in our QA pipeline. Our system performs comparably to the state of the art in the open-domain multiple choice question answering field (Chaturvedi et al. 2018), which is able to achieve accuracy of 35.8% over 4 possible questions on the TQA dataset—rather than the 3 in *HQ*. Normalising this to 3, this gives an estimated accuracy between 42.9% and 50.0%. Although, the differing datasets make this difficult to draw an exact comparison between these two systems; as no prior

papers explore *HQ* specifically, there is no exact basis for comparison, but we can still conclude it performs notably for an open-domain QA system.

We also find that our system is fast enough to be used during live games of *HQ*. The average duration for the full final pipeline takes 9.4s on test hardware; trivially reduced with more aggressive compute hardware.

B System Limitations

While we note that our pipeline performs exceptionally well for the answering of simple questions (of which there is clear and concise answer available from our corpora), complex questions are vastly more challenging; for example:

Which does Stan Lee NOT do in a Marvel movie cameo?
"Mow a lawn", "Give a haircut", "Ride a bus"

Information relating to *such* a specific event is much harder to condense into a concise passage. We must find a piece of text that unambiguously states or insinuates at the correct answer to be able to answer this with a DNN; very challenging for questions such as this, where many facts must be taken into context all at once. This is theoretically possible using a BERT model fine-tuned for such a task, but requires a training dataset at least as complex as the possible questions that will be asked (Goodfellow et al. 2016). We are also highly dependent on the article retrieval phase obtaining all the facts relevant to such a question. If a much larger number of sources were considered and these results combined into a consistent passage, answers to questions of this class could be generated. Questions of the complex nature allow naive answer counting techniques using a search engine (Schwartz 2018) to perform favourably due to considering all possible indexed search results relating to a few keywords, and evaluating which terms are mentioned more; thus determining which is likely to be correct.

We noted during passage retrieval that a fine-tuned BERT model performing MS MARCO passage ranking or QNLI entailment classification degraded overall performance when used alongside our algorithmic ranker implementing BM25F. This is due to BM25F retrieving only based on the keywords in a given passage and not the underlying meaning. The conflict of the deep language modelling and keyword exact-matching sometimes lead to no passages being returned at all. Ideally, only a DNN passage ranker would be used for improved performance, but this resulted in unusably slow performance for our application. However, the fundamental problem was that the content needed to answer questions was often not being retrieved as required. Even using Wikipedia and Contextual Web Search, information relating to some topics was hard to find for very contemporary questions or where the extracted keywords were not optimal. If the project were to be repeated, greater emphasis would be placed on the construction of high quality search queries, as well as the utilisation of more corpora.

Being limited to a single GPGPU for DNN training during this project (due to resource limitations) prevented us from using the larger and more performant $\text{BERT}_{\text{LARGE}}$ model for our trained DNNs, measurably limiting potential performance during answer inference (Devlin et al. 2018).

C Organisation and Management

Our approach of first creating an entire baseline system, including open-ended answer inference and algorithmic passage ranking, allowed us to determine which of the more technically complex

additions to the system was worthwhile and how this improved or impacted performance. Adhering to our Agile development methodology, we were able to adapt and re-order requirements of the system as such discoveries were made, helping to ensure maximal performance was achieved for the final system as efficiently as possible.

Using high quality, large datasets to train our final answer inference system ensured that the system achieves the highest possible performance without requiring large amounts of time to compile such datasets ourselves. The decision to use PyTorch as the deep learning framework was wise, due to increased simplicity presented in the API compared to competing libraries. While this did cause friction when trying to interoperate with the pre-fine-tuned MS MARCO passage ranking model (trained using TensorFlow) (Nogueira & Cho 2019), the time saved during development compensates favourably for this.

If additional time was available, it would be worth exploring the effect of increasing the number of corpora used to query from; the lack of relevant passages was the main bottleneck in the system's overall accuracy. Additionally, as this project focused on optimising many stages of a single pipeline, there was insufficient time to maximally optimise any given stage; this is the greatest issue in the design of the project. If the project were to be repeated, research would be more focused onto a particular stage of the pipeline and would use accepted, unmodified, state of the art implementations for the other stages.

VI CONCLUSIONS

Over the course of this project, we have explored a number of different methods and models in order to effectively determine if we can reliably answer questions posed by *HQ Trivia*. We are able to achieve exact match accuracy of 52% when answering questions sampled directly from *HQ Trivia*. While not reliable for answering questions, this is comparable to the state of the art in open-domain, multiple-choice QA (Chaturvedi et al. 2018). However, exact comparisons are not able to be explicitly made due to no prior papers presenting accuracy results for *HQ Trivia* specifically; classes and difficulty of questions are not commutative between datasets.

We verify research into language modelling for DNNs, confirming that these provide excellent results for passage ranking and question answering tasks. We find that using a limited number of potential corpora results in poor retrieval accuracy when searching for relevant textual passages, creating a bottleneck for the entire QA system.

Future work could explore the degree to which different sources improve this retrieval accuracy; such as having an inverted indexed offline dump of Wikipedia and avoiding Wikipedia's low performance built in API search (Chen et al. 2017). This could also factor in techniques from naive attempts to beat *HQ*, such as search engine result counting approaches—effective in the answering of some particular questions (Schwartz 2018). Retrieval would also benefit from research into the most effective manner to create queries from a question for a particular corpus, potentially using DNNs rather than the algorithmic approach taken in this paper. Additionally, as we were constrained by resources available for DNN training, it would be trivial to see how the $\text{BERT}_{\text{LARGE}}$ model impacts DNN performance. Finally, it would be compelling to explore how the effect of newer, larger and more complex QA datasets impact the performance of the answer inference stage. One such dataset is Natural Questions (Kwiatkowski et al. 2019); the current state of the art F1 scores only reach 53% for finding short answers to questions with a BERT based model.

REFERENCES

Chaturvedi, A., Pandit, O. & Garain, U. (2018), CNN for text-based multiple choice question answering, *in* 'Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)', Association for Computational Linguistics, Melbourne, Australia, pp. 272–277.

Chen, D., Fisch, A., Weston, J. & Bordes, A. (2017), 'Reading wikipedia to answer open-domain questions', *CoRR* **abs/1704.00051**.

Devlin, J., Chang, M., Lee, K. & Toutanova, K. (2018), 'BERT: pre-training of deep bidirectional transformers for language understanding', *CoRR* **abs/1810.04805**.

Goodfellow, I., Bengio, Y. & Courville, A. (2016), *Deep Learning*, The MIT Press.

Gotterbarn, D., Miller, K. & Rogerson, S. (1997), 'Software engineering code of ethics.', *Commun. ACM* **40**, 110–118.

Green, Jr., B. F., Wolf, A. K., Chomsky, C. & Laughery, K. (1961), Baseball: An automatic question-answerer, *in* 'Papers Presented at the May 9-11, 1961, Western Joint IRE-AIEE-ACM Computer Conference', IRE-AIEE-ACM '61 (Western), ACM, New York, NY, USA, pp. 219–224.

Hochreiter, S. & Schmidhuber, J. (1997), 'Long short-term memory', *Neural Comput.* **9**(8), 1735–1780.

Joshi, M., Choi, E., Weld, D. S. & Zettlemoyer, L. (2017), 'TriviaQA: A large scale distantly supervised challenge dataset for reading comprehension', *CoRR* **abs/1705.03551**.

Józefowicz, R., Vinyals, O., Schuster, M., Shazeer, N. & Wu, Y. (2016), 'Exploring the limits of language modeling', *CoRR* **abs/1602.02410**.

Kwiatkowski, T., Palomaki, J., Redfield, O., Collins, M., Parikh, A., Alberti, C., Epstein, D., Polosukhin, I., Kelcey, M., Devlin, J., Lee, K., Toutanova, K. N., Jones, L., Chang, M.-W., Dai, A., Uszkoreit, J., Le, Q. & Petrov, S. (2019), 'Natural questions: a benchmark for question answering research', *Transactions of the Association of Computational Linguistics* .

Lai, G., Xie, Q., Liu, H., Yang, Y. & Hovy, E. (2017), 'RACE: Large-scale reading comprehension dataset from examinations', *arXiv preprint arXiv:1704.04683* .

Mitra, B. & Craswell, N. (2017), 'Neural models for information retrieval', *CoRR* **abs/1705.01509**.

Nguyen, T., Rosenberg, M., Song, X., Gao, J., Tiwary, S., Majumder, R. & Deng, L. (2016), 'MS MARCO: A human generated machine reading comprehension dataset', *CoRR* **abs/1611.09268**.

Nogueira, R. & Cho, K. (2019), 'Passage re-ranking with BERT', *CoRR* **abs/1901.04085**.

Pan, X., Sun, K., Yu, D., Ji, H. & Yu, D. (2019), 'Improving question answering with external knowledge', *CoRR* **abs/1902.00993**.

Paszke, A., Gross, S., Chintala, S. & Chanan, G. (2016), 'PyTorch: An open source deep learning platform [...]', https://pytorch.org. [Online, Accessed: 2018-10-30].

Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K. & Zettlemoyer, L. (2018), 'Deep contextualized word representations', *CoRR* **abs/1802.05365**.

Radford, A., Narasimhan, K., Salimans, T. & Sutskever, I. (2018), 'Improving language understanding by generative pre-training'. Technical Report, OpenAI.

Rajpurkar, P., Zhang, J., Lopyrev, K. & Liang, P. (2016), 'SQuAD: 100, 000+ questions for machine comprehension of text', *CoRR* **abs/1606.05250**.

Robertson, S. E. & Walker, S. (1994), Some simple effective approximations to the 2-poisson model for probabilistic weighted retrieval, *in* 'Proceedings of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval', SIGIR '94, Springer-Verlag New York, Inc., New York, NY, USA, pp. 232–241.

Schwartz, B. (2018), 'Building the ultimate HQ bot', https://medium.com/@LtHummus/building-the-ultimate-hq-bot-c8f89a120fe2. [Online, Accessed: 2018-10-20].

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. & Polosukhin, I. (2017), 'Attention is all you need', *CoRR* **abs/1706.03762**.

Wang, A., Singh, A., Michael, J., Hill, F., Levy, O. & Bowman, S. R. (2018), 'GLUE: A multi-task benchmark and analysis platform for natural language understanding', *CoRR* **abs/1804.07461**.

Yang, W., Xie, Y., Lin, A., Li, X., Tan, L., Xiong, K., Li, M. & Lin, J. (2019), 'End-to-end open-domain question answering with BERTserini', *CoRR* **abs/1902.01718**.