MACHINE READING COMPREHENSION APPLICATION FOR CNN NEWS ARTICLES

COS80023 BIG DATA – D/HD PROJECT

KIEN QUOC MAI
103532920

TABLE OF CONTENTS

I. INTRODUCTION

Recently, Large Language Model (LLM) has gained significant popularity with the wide adoption of OpenAI's ChatGPT and similar models. LLM is especially useful as it allows us to process, transform and extract valuable information from unstructured data in the form of natural language, which is the primary communication method between humans. Some of its applications include but not limited to text generation, text summarisation, sentimental analysis and question answering. This project will explore the used of LLMs for machine reading comprehension application.

Machine reading comprehension (MRC) is a subset of Question answering (QA). The input for MRC consists of 2 pieces of text: a question and a context. The task for the model is to read and comprehend the given context, then extract the answer to the given question from that specific context. As opposed to general QA, MRC focus more on the ability of a model to understand the relationship between a context and a question in order to locate the correct answer instead generating a response based on the given inputs.

The goal of this project is to develop an application that utilises LLMs to answer questions about news articles. The dataset chosen for this purpose is the NewsQA dataset from Microsoft research. This dataset contains more than 100,000 question and answer pairs on more than 12,000 news articles from CNN. In addition, those questions require human-level comprehension and reasoning skills, which cannot be achieve using simple model or algorithm.

II. INFORMATION RETRIEVAL MODEL

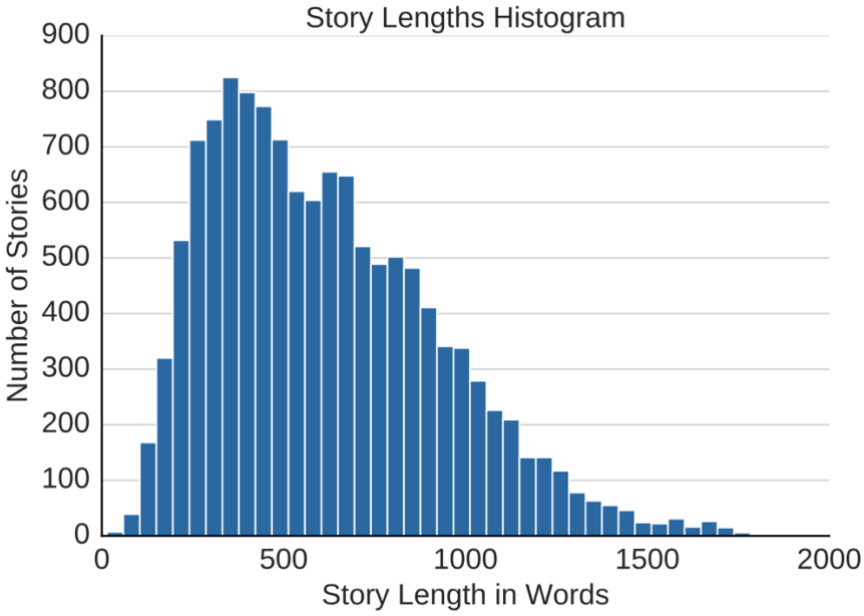

FIGURE 1. STORY LENGTH DISTRIBUTION OF NEWSQA

As demonstrated by Figure 1. story length distribution of nEWSQA, news articles in the NewsQA dataset are quite long with up to over 1500 words. This means a single article

cannot be passed as an input of an LLM since they have a limit on maximum input tokens (the Flan-T5 model used in this project only has a maximum token limit of 512). To resolve this problem, the data need to be processed using truncation or information retrieval to reduce the length and extract appropriate information.

A. TRUNCATION

Truncation refers to the process of truncating text that exceed the token limit from the end. This is the simplest solution as it just sacrifice some part of the information to fit the token limit. As a result, important information might be discarded which make the question unanswerable. However, this method still produces very good accuracy. After being truncated, the 91.4% of input contexts in the test dataset still contain the correct answers.

B. INFORMATION RETRIEVAL

Information retrieval refers to the process of splitting the context into multiple smaller parts called chunks and only keeping chunks that are relevant to the question. In this project, each news article is split into chunks of at least 2 sentences with a desired length in words. In addition, there are overlaps of at least 1 sentence and at most some percentages of the desired chunk length between consecutive chunks in order to preserve the semantic meanings. After that, relevant chunks are ranked in terms of relevancy using a pretrained cross-encoder model ms-marco-MiniLM-L-6-v2 before being concatenated and truncated to 512 tokens in length. The cross-encoder model takes 2 pieces of text, which are a question and a chunk, and generate a number between 0 and 1 indicating the relevancy between them. The pretrained model ms-marco-MiniLM-L-6-v2 was trained on the MS Marco Passage Ranking task and provide excellent performance without any finetuning.

Several experiments are conducted to find the optimal chunk's length and overlapping percentages.

TABLE 1. CHUNKING METHOD EXPERIMENTS

Method	Accuracy on validation set	Accuracy on test set
Simple truncation	92.40	91.41
100 words with 20% overlap	95.67	96.20
100 words with 10% overlap	**95.67**	**96.20**
75 words with 15% overlap	95.21	95.23
50 words with 10% overlap	95.19	94.90

The results of the experiments suggest that splitting news articles into chunks of around 100 words with an overlap of 10% produce the best outcome with 96.2% accuracy. This means 96.2% of the truncated input contexts in the test dataset still contain the correct answers.

III. MRC MODEL

After processing the data, a large language model (LLM) can be applied to perform the MRC task. In simple terms, LLMs are machine learning models that are capable of predicting the next words in a sequence. They are often trained on large corpus of human generated text which allow it to understand and imitate human language. To apply an LLM to MRC task, a question and a context are composed into prompts with the format: “question: <question> answer: <answer>”. Then, the prompt is converted into tokens via the use of a tokenizer. After that, the tokenised input is fed into the model to generate the answer in the form of tokens. Those tokens need to be decoded back into natural language.

```
def inference(ds):
    qa_input = [
        f"question: {ds['question'][idx]} " f"context: {ds['context'][idx]}"
        for idx in range(len(ds["question"]))
    ]
    inputs = tokenizer(
        qa_input,
        return_tensors="pt",
        truncation=True,
        padding=True,
        max_length=max_length,
    ).to(torch.device("cuda"))
    outputs = model.generate(
        input_ids=inputs["input_ids"],
        attention_mask=inputs["attention_mask"],
        do_sample=False,
        max_new_tokens=max_target_length
    )
    ds["output"] = tokenizer.batch_decode(outputs, skip_special_tokens=True)
    return ds

model.eval()
with torch.no_grad():
    predicted_result = dataset["test"].map(inference, batched=True, batch_size=32)
```

FIGURE 2. APPLY LLM FOR MRC

A. MODEL SELECTION

As LLMs are extremely large in scale, some having millions or even billions of parameters, it is nearly impossible to develop and train a LLM from scratch with limited resources. Fortunately, many pretrained LLMs have been made freely available and open sourced. One can leverage the power of those pretrained models and apply them to specific use cases. For this project, the pretrained model chosen for experimenting was the Google’s Flan-T5. This is an instruction-tuned version of the T5 model meaning it has been trained on instructional datasets for various different tasks. Hence, it has better ability to follow instructions compared to other LLMs, which is generally only good at text generation. The second choice needed to be made was which options of the Flan-T5 to use as it comes in many sizes ranging from around 300 million parameters for the base option to around 11 billion parameters for the XL option. Because the machine used for this project has limited hardware resources, the only viable options were the base and the large version of the Flan-T5 model. Between them,

the Flan-T5 large was chosen as it has larger number of parameters which lead to better performance in theory.

B. FINETUNING

Since Flan-T5 is a general purpose LLM, it needs to be finetuned to adapt to the new domain, which is the domain of news articles. However, it is impractical to fully finetune the model since training LLMs is extremely resource intensive. The Flan-T5-large model has around 700 million parameters which would require a lot of memory and could takes days to train on consumer hardware. Fortunately, LLMs can be finetuned using Parameter-Efficient Fine-Tuning (PEFT) that drastically reduce the computational power and training time. The idea behind PEFT is actually very simple. Firstly, the pretrained model is frozen. Then, additional layers are added. While finetuning, only the additional layers are modified while the frozen layers are left untouched. Hence, it requires less computational power and training time since only a small number of parameters need to be optimised.

In this project, the particular PEFT technique used to finetune the Flan-T5 model is Infused Adapter by Inhibiting and Amplifying Inner Activations (IA^3). The IA^3 method introduces small trainable vectors into different components of a Transformer model. Those vectors are used to perform element-wise rescaling of inner model activations. For example, a layer of the form $h=\mathrm{Wx}$ is transformed into $h= \mathrm{l_w} \odot \mathrm{Wx}$ by multiplying a vector $\mathrm{l_w}$ in an element-wise manner after broadcasting it to the shape of W.

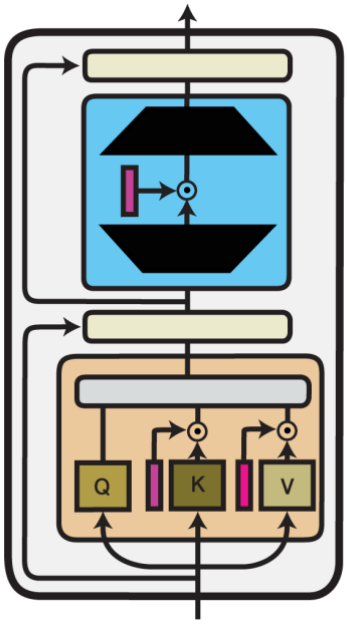

FIGURE 3. IA^3 ARCHITECTURE (PARTS SHADED IN MAGENTA ARE TRAINABLE VECTORS)

In this project, trainable vectors are injected into self-attention modules “query” (q), “value” (v) and a dense feed-forward layer “wo”. This is achieved with the help of the Hugging Face’s PEFT library. The library facilitates the finetuning process by providing the implementation of various popular PEFT techniques including IA^3. The configuration is done as demonstrated in

Figure 4. IA^3 configuration using PEFT library After configuring the IA^3 vectors, the total number of trainable parameters is only 282,624 which is a fraction of the pretrained Flan-T5 model with over 700 billion parameters.

```python
from transformers import AutoModelForSeq2SeqLM, AutoTokenizer
from peft import PeftModelForSeq2SeqLM, get_peft_config

config = {
    "peft_type": "IA3",
    "task_type": "SEQ_2_SEQ_LM",
    "inference_mode": False,
    "target_modules": ["q", "v", "wo"],
    "feedforward_modules": ["wo"],
}

peft_config = get_peft_config(config)
tokenizer = AutoTokenizer.from_pretrained("google/flan-t5-large")
model = AutoModelForSeq2SeqLM.from_pretrained("google/flan-t5-large")
peft_model = PeftModelForSeq2SeqLM(model, peft_config)
```

FIGURE 4. IA^3 CONFIGURATION USING PEFT LIBRARY

After that, the model can be trained effortlessly using the Trainer class provide by the Hugging Face Transformer library. First, hyperparameters need to be defined as training arguments. Some of the most important hyperparameters are learning rate, number of training epochs, and batch size.

```python
from transformers import (
    DataCollatorForSeq2Seq,
    Seq2SeqTrainingArguments,
    Seq2SeqTrainer,
)

epochs = 2
lr = 4e-3
steps = 5000

training_args = Seq2SeqTrainingArguments(
    output_dir=f"./{checkpoint}",
    learning_rate=lr,
    auto_find_batch_size=True,
    num_train_epochs=epochs,
    warmup_ratio=0.05,
    weight_decay=0.01,
    lr_scheduler_type="linear",
    save_strategy="steps",
    save_steps=steps,
    logging_steps=steps,
    logging_first_step=True,
    evaluation_strategy="steps",
    eval_steps=steps,
    predict_with_generate=True,
    generation_max_length=max_target_length,
)
```

FIGURE 5. TRAINING ARGUMENTS CONFIGURATION

The next step is to configure the Trainer class by passing the aforementioned arguments along with other components such as the dataset, the tokenizer and the data collator, which help forming batches of data. Finally, the training process can be started by calling the train function. This function effectively abstracts the conventional training loop in machine learning.

```python
data_collator = DataCollatorForSeq2Seq(
    tokenizer=tokenizer,
    model=peft_model,
    padding="max_length",
    max_length=max_length,
    return_tensors="pt",
)
trainer = Seq2SeqTrainer(
    model=peft_model,
    args=training_args,
    train_dataset=seq2seq_dataset["train"].with_format("torch"),
    eval_dataset=seq2seq_dataset["test"].with_format("torch"),
    data_collator=data_collator,
    tokenizer=tokenizer,
    compute_metrics=compute_metrics,
)

trainer.train()
```

FIGURE 6. TRAINER CONFIGURATIONS

IV. ANALYSIS

A. EVALUATION METRICS

To evaluate the results of the model on the MRC task, 3 different metrics are used: Exact match (EM), F1, and BLEU. First of all, EM considers an answer correct if it matches exactly word by word to the ground truth. Although it is an intuitive metric and show how well the model produce correct answers, it is a little too extreme for a lot of cases because if the model misses a single word in an answer, that answer will be considered as incorrect. The F1 metric is used to yield more insights by considering individual word matches. It calculates the harmonic mean of precision and recall for each word in the generated answer compared to the ground truth. As a result, F1 score introduces a spectrum of correctness instead of being binary as in EM. However, since F1 score does not consider the order of words, it often fails to capture the actual semantic meaning of an answer. For example, given a ground truth of "Paris is the capital of France" and the generated answer "France is the capital of Paris", the generated answer would score a perfect 100 for F1 as every word in it appears in the ground truth. To mitigate this, BLEU score is used as it consider variable length phrase matches (n-grams) instead of individual word matches against the ground truth. The above example evaluating with BLUE score produces a result of only 53%. Together, all those 3 metrics provide valuable insights on the performance of an MRC model.

B. RESULTS

TABLE 2. MRC FINETUNING RESULTS

Method	Trainable parameters	EM	F1	BLEU
Human baseline	-	46.50	69.40	56.00
Pretrained	-	30.51	45.69	14.66
IA^3 finetuned	282,624 (0.0361%)	56.60	71.28	55.36

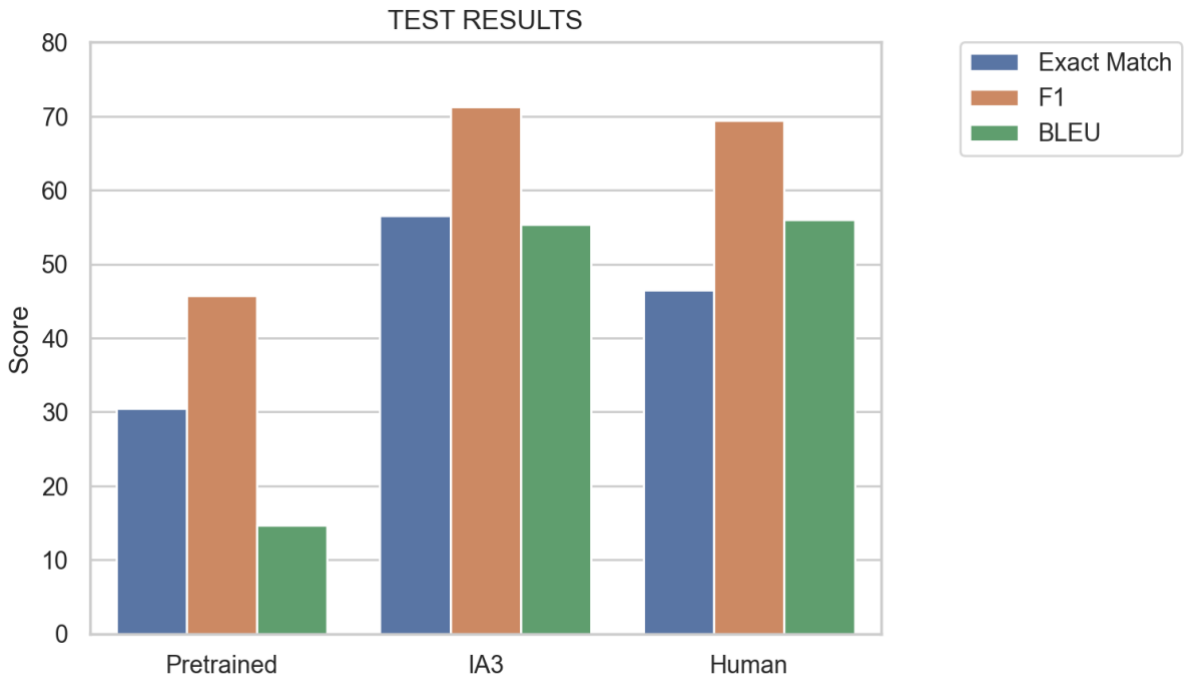

FIGURE 7. MRC FINETUNING RESULTS BAR CHART

The pretrained Flan-T5 model achieves decent scores of around 30 for EM and 45 for F1. However, those are far from the human baseline at around 46 for EM and 69 for F1, as stated by Trischler et al (2016). Finetuning the model using IA^3 technique successfully bridge that gap by significantly improve the performance of the MRC model over all 3 metrics. For EM, there is a 85% improvement from 30 to nearly 57, while F1 score sees an increase of 56% from 45 to over 71. In general, these impressive results are comparable to that of human and they seem to surpass human baseline in terms of EM. Note that since the human baseline is evaluated against a subset of 1000 samples from the test set (Trischler et al, 2016), these are only rough comparisons for referencing.

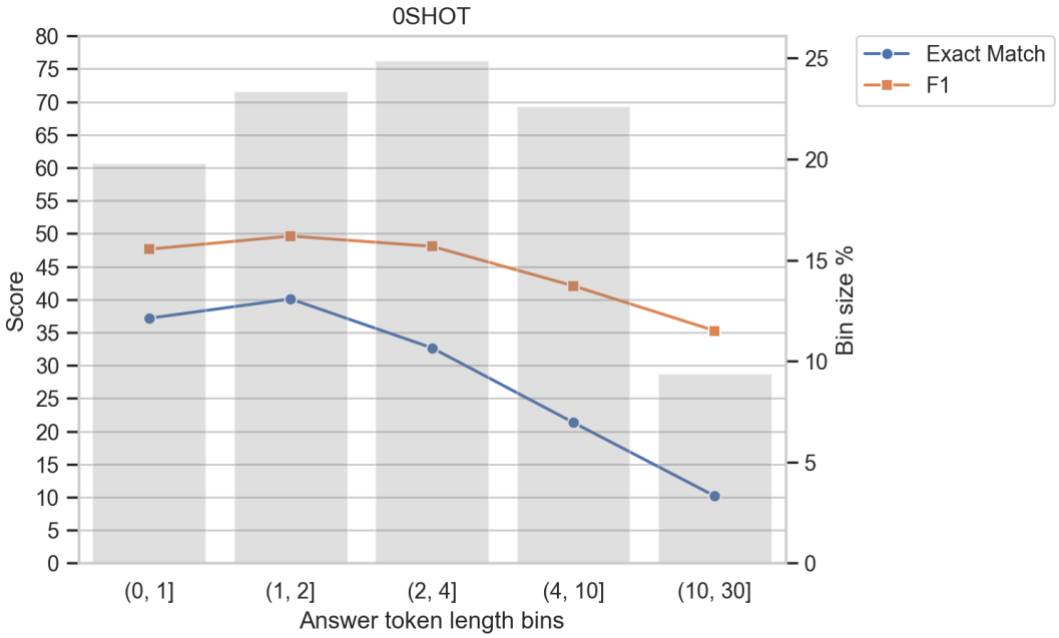

FIGURE 8. 0-SHOT RESULT FOR DIFFERENT ANSWER LENGTHS

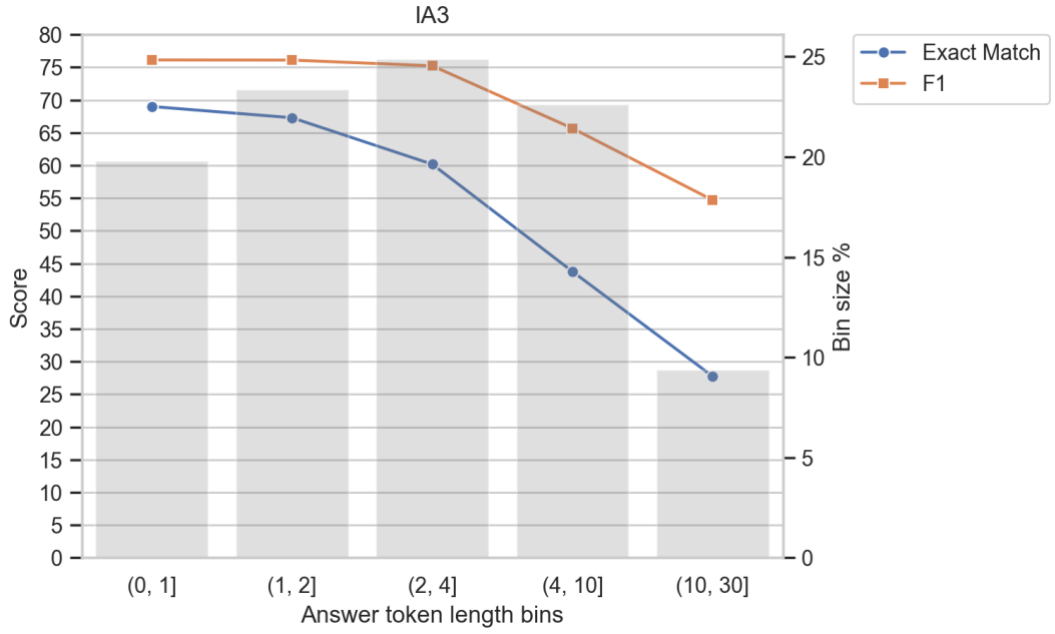

FIGURE 9. IA³ RESULT FOR DIFFERENT ANSWER LENGTHS

To acquire more insights from the results, the test dataset is categorised into different subsets based on the answer length. As highlighted by Figure 8. 0-shot Result for different answer lengths and Figure 9. IA^3 Result for different answer lengths, although there is a clear improvement over the board, the performance pattern remains unchanged. As the answer gets longer, the performance of both the pretrained and finetuned models decrease in terms of both EM and F1. It is easily to observed that the result for long answers that are above 10 words is significantly lower than shorter answers. However, since they only account for less than 10% of the test set, the impact on the overall performance is not that drastic.

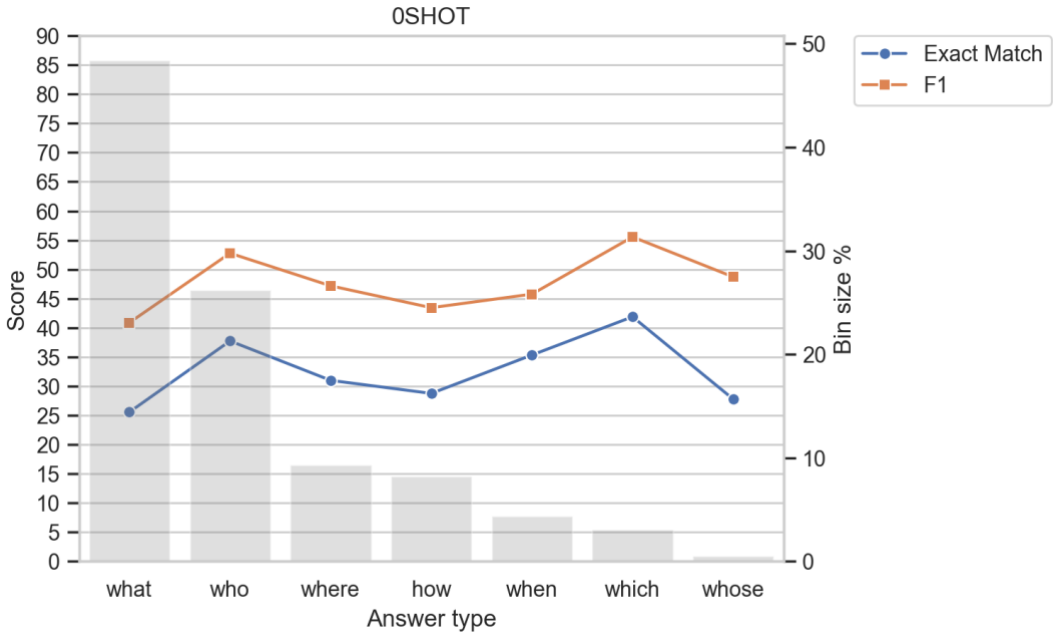

FIGURE 10. 0-SHOT RESULT FOR DIFFERENT ANSWER TYPES

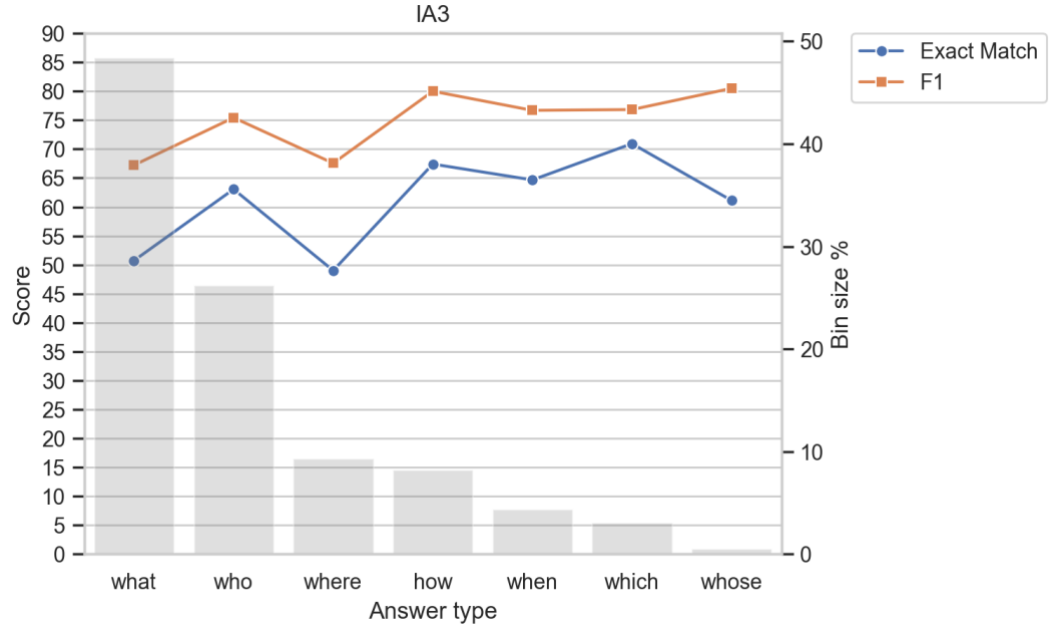

FIGURE 11. IA^3 RESULT FOR DIFFERENT ANSWER TYPES

Similarly, the test set is categorised into subsets of different question types to examine whether they affect the MRC model performance. **Error! Reference source not found.** and Figure 11 suggest that question types do not seem to affect the performance. Both EM and F1 scores are fairly consistent across different question types with some minor fluctuations. Despite that, the figures highlight a small but interesting difference between the performance pattern of pretrained and finetuned models. The finetuned model performs nearly the best on “how” questions compared to other types, whereas that of the pretrained model’s is

among the worst. However, that difference do not make a significant impact as “how” questions only accounts for approximately 10% of the test set.

V. CONCLUSION

To summarise, this project has extensively investigated and implemented a machine reading comprehension application (MRC) for news articles utilising the power of large language models (LLM). Overall, the application and the MRC model in specific achieve quite impressive performance, which is comparable to human baseline, despite being finetuned with such small number of trainable parameters (0.036% of the pretrained model) on consumer hardware. As a result, it allows effective information extraction from news article which is widely beneficial for a lot of use cases from simple chat bots to natural language processing pipelines (NLP). All in all, this project has successfully demonstrated the vital role of LLMs in NLP and the ease of integrating and adapting them to specific needs.

VI. REFERENCES

Adapter Methods — adapter-transformers documentation n.d., viewed 1 November 2023, <https://docs.adapterhub.ml/methods.html>.

Liu, H, Tam, D, Muqeeth, M, Mohta, J, Huang, T, Bansal, M & Raffel, C 2022, *Few-Shot Parameter-Efficient Fine-Tuning is Better and Cheaper than In-Context Learning*, viewed 4 November 2023, <https://arxiv.org/abs/2205.05638>.

‘NewsQA Dataset’ 2019, *Microsoft Research*, viewed 4 November 2023, <https://www.microsoft.com/en-us/research/project/newsqa-dataset/>.

PEFT n.d., viewed 2 November 2023, <https://huggingface.co/docs/peft/index>.

Trischler, A, Wang, T, Yuan, X, Harris, J, Sordoni, A, Bachman, P & Suleman, K 2016, *NewsQA: A Machine Comprehension Dataset*, viewed 4 November 2023, <https://arxiv.org/abs/1611.09830>.

VII. APPENDIX

All the codes for this project are in the form of Jupyter notebooks. Link to the GitHub repo: https://github.com/Legacy107/COS80023-NewsQA-MRC