Advanced Generative Chatbot Design

Eric Barnes Massimiliano Repupilli Jonathan Agustin

University of San Diego

Abstract

This study explores the creation of open source conversational chatbots, capitalizing on transformer networks and the nuanced application of transfer learning from pre-trained models. We unpack the methods, experimental processes, outcomes, and nuanced analyses involved in fine-tuning BERT, DistilBERT, and RoBERTa with version 2 of the Stanford Question Answering Dataset (SQuAD). We find that RoBERTa outperforms its counterparts, which highlights the importance of optimized pretraining procedures. Overall, all three models were consistently challenged when handling unanswerable questions, indicating an area for future exploration.

1 Introduction

AI constantly faces the challenge of creating chatbots that can effectively mimic human interaction across various topics. Despite advances in deep learning, understanding language and managing dialogues remain complex. Our project focuses on developing an engaging chatbot for a wide range of subjects, using Open Source NLP techniques. We leverage transfer learning from pretrained models like BERT. This exploration chronicles our process, including literature review, data collection, model design, training methods, evaluation, and analysis. We compare different three transformer models and report statistics. Ultimately, we aim to contribute to future research in conversational AI systems.

2 Related Work

BERT Devlin et al. (2018) advances NLP through introducing a way to learn from before and after a position within a text, which was not done previously. This bidirectional approach enhanced BERT's adaptability, making it a top solution for many tasks with minimal adjustment. Subsequent studies concentrate on optimizing BERT's pretraining methods. RoBERTa emphasized the importance of parameters like batch size, training duration, and dataset size. DistilBERT streamlined the essence of BERT, achieving a compact model without compromising on performance, bridging the gap between computational efficiency and effectiveness.

3 Exploration

3.1 SQuAD

The Stanford Question Answering Dataset (SQuAD) is a popular dataset for training Q&A models. We use the second version of SQuAD, which improves upon its previous version

through adding unanswerable questions. This impossibility challenges a model's ability to understand and interpret text. We aim to advance these transformer technologies through leveraging existing pretrained models, datasets, and distillations.
SQuAD 2.0 challenges a model's ability to understand and interpret text.

Table 1: Overview of the SQuADv2 Dataset

Detail	**Information or Statistics**
Dataset Name	squad_v2
Description	Combines 100,000 questions with over 50,000 unanswerable questions written adversarially by crowdworkers to look similar to answerable ones. Systems must determine when no answer is supported by the paragraph and abstain from answering.
Citation	*Rajpurkar, Pranav et al. SQuAD: 100,000+ Questions for Machine Comprehension of Text, arXiv e-prints, 2016.*
Homepage	SQuAD Explorer
Train Examples	130,319
Validation Examples	11,873

Table 1 provides details about the SQuAD 2.0 dataset. Unanswerable questions tests not only information retrieval skills but also discernment in recognizing when text does not contain an answer.

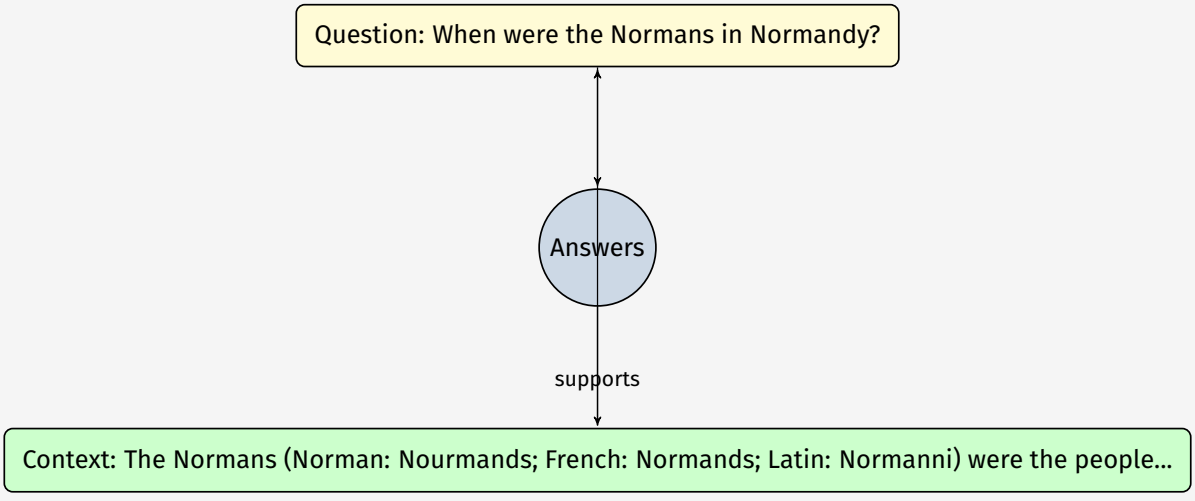

Figure 1: Diagram representing the relationship between question, answer, and context.

Figure 1 shows the steps involved in answering a question. The model is limited to only the context when answering a question.

3.2 BERT

BERT (Bidirectional Encoder Representations from Transformers) significantly advanced the field of natural language understanding with its innovative approach to contextual encoding. This model, introduced by Devlin et al. (2018), allows for highly effective transfer learning, particularly in the realm of language comprehension tasks. We explore the innovative bidirectional nature of this model and measure its effectiveness.

3.3 DistilBERT

Sanh et al. (2019) introduced DistilBERT, a distilled version of the original BERT model that retains most of its performance characteristics with significantly fewer resources. Our implementation involved fine-tuning and exploring its efficient architecture, which is ideal for resource-restricted environments.

3.4 RoBERTa

RoBERTa, Liu et al. (2019), refines pre-training process through, for example, modifying the masking pattern applied to training data and through extending the training period. Our implementation of RoBERTa explored its versatility in generating contextually appropriate multilingual responses.

4 Model Development

Our strategy leveraged the HuggingFace Model Hub and other third-party libraries. Three pretrained models—BERT, DistilBERT, and RoBERTa—were inputted into an encoder-decoder pipline. Initialization with pretrained weights was important as it allowed models to capitalize on previously acquired training activitys. The models were trained using A100 and T4 GPUs on Google Colab. This cloud setup enabled remote development at little to no cost. Persistence between sessions was achieved through Google Drive, which allowed us to save and load checkpoints.

5 Training

Handling comprehensive datasets like SQuAD in NLP tasks require preprocessing, including tokenizing sentences, adjusting sequence lengths, and calculating indices. The process is often time-consuming and susceptible to errors. We streamlined training with using a third-party library called `txtai` (see code in the Appendix). At the beginning, we

manual preprocessed the datasets, but it was difficult. Importing the library allowed for more concise code and allowed us to go to the next step without ad-hoc trial-and-error data manipulation.

6 Evaluation

After training a model, we used the `evaluate` library from HuggingFace's Transformer library. We used the official SQuADv2 to measure model predictions. The results were outputted, saved, and published. Despite the need to handle unanswerable questions, the evaluation phase provided crucial insight in improving overall performance.

7 Results

RoBERTa outperformed its counterparts, which highlights the importance of optimized pretraining procedures. Models were consistently challenged when handling unanswerable questions, indicating an area for future exploration.

- GitHub

  https://github.com/aai520-group6/project

- BERT

  https://huggingface.co/aai520-group6/bert-finetuned-uncased-squad_v2

- DistilBERT

  https://huggingface.co/aai520-group6/distilbert-finetuned-uncased-squad_v2

- RoBERTa

  https://huggingface.co/aai520-group6/roberta-finetuned-uncased-squad_v2

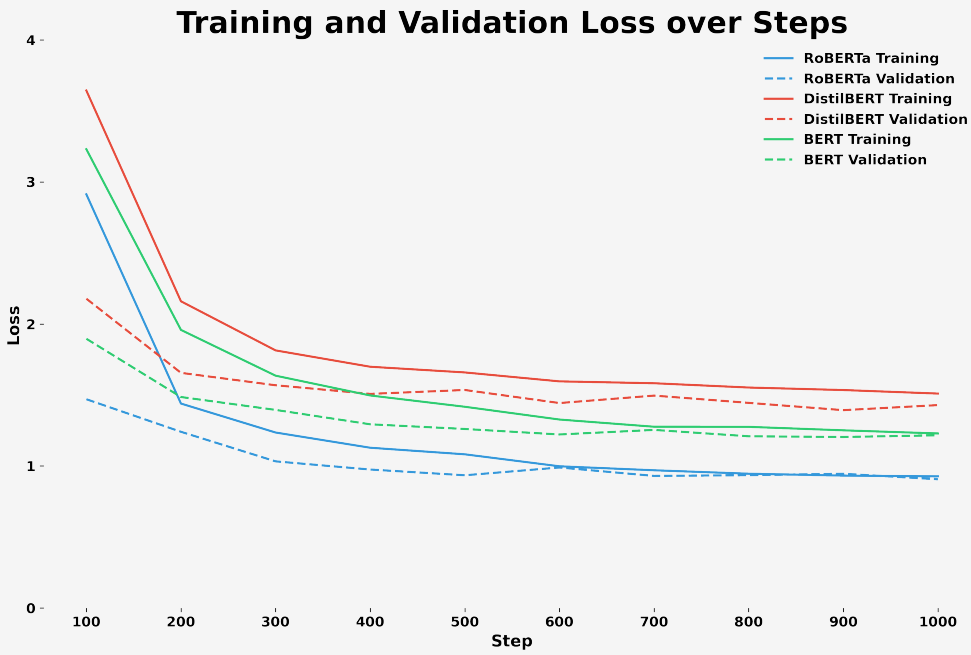

Figure 2: Visualization of model loss over time during training.

Fine-tuned Model Comparison			
Metric	BERT	DistilBERT	RoBERTa
Total Examples	11,873	11,873	11,873
Exact Match	27.878	23.347	37.404
F1 Score	32.130	26.870	40.428
Total with Answer	5,928	5,928	5,928
Has Answer Exact Match	50.405	38.630	72.588
Has Answer F1 Score	58.920	45.686	78.644
Total without Answer	5,945	5,945	5,945
No Answer Exact Match	5.416	8.108	2.321
No Answer F1 Score	5.416	8.108	2.321
Best Exact Match	50.114	50.114	50.097
Best F1 Score	50.114	50.114	50.097

8 Discussion

Our exploration shows the relative ease of transfer learning in dialog-oriented tasks. RoBERTa's superior performance shows the importance of nuanced training and specific selections of hyperparameters like dataset size and training duration. The models were able to answer questions with high accuracy, but struggled with unanswerable questions. Future iterations should emphasize training with unanswerable questions to enhance a

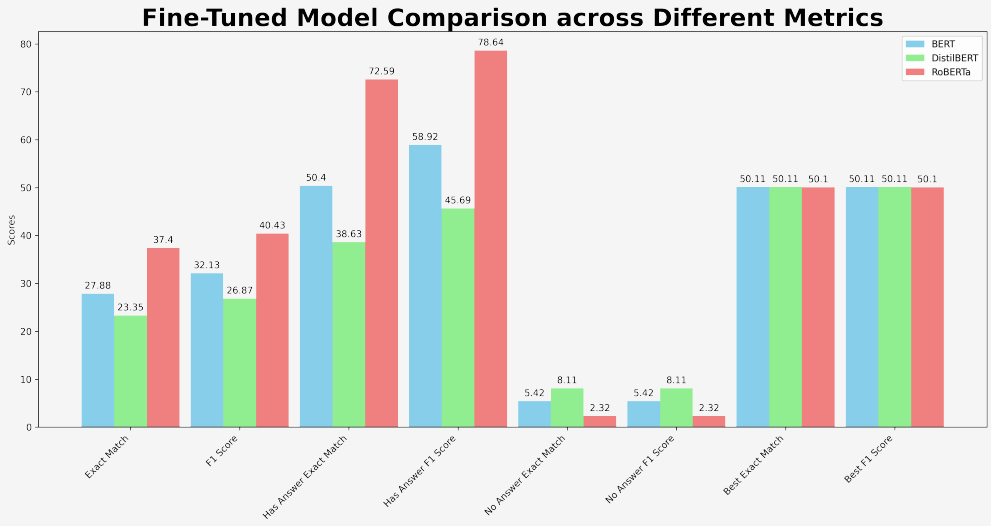

Figure 3: Comparison of fine-tuned models on various metrics.

model's discernment abilities. Solving these issues will allow for conversational agents to rise to the level of for real-world human interaction.

8.1 Challenges and Limitations

We faced Out-of-Memory (OOM) issues using Google Colab. These issues often required us to adjust our approach, such as using smaller data batches or simplifying model structures. Despite its advanced features, Colab sometimes had performance limitations, and session management was disruptive, causing delays in our workflow. The interactive terminal in Colab was often slow, especially through a VPN. The speed was good enough for notebooks and we often used the notebook to invoke terminal commands. This slowed down testing and debugging, but we were able to work around this limitation.

9 Conclusion

We explored conversational chatbots, fine-tuning transformer networks to enhance a multi-turn dialog capability. Our exploration and analysis is all published to open source, guiding future endeavors towards refining conversational AI technologies.

References

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., & Stoyanov, V. (2019). Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Rajpurkar, P., Jia, R., & Liang, P. (2018). Know what you don't know: Unanswerable questions for squad. *arXiv preprint arXiv:1806.03822*.

Rajpurkar, P., Zhang, J., Lopyrev, K., & Liang, P. (2016). SQuAD: 100,000+ Questions for Machine Comprehension of Text. *arXiv e-prints*, Article arXiv:1606.05250, arXiv:1606.05250.

Sanh, V., Debut, L., Chaumond, J., & Wolf, T. (2019). Distilbert, a distilled version of bert: Smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, *30*.

A Appendix

A.1 Training Hyperparameters

Hyperparameter	**Value**
Learning Rate	2e-05
Train Batch Size	64
Eval Batch Size	64
Gradient Accumulation Steps	4
Total Train Batch Size (effective)	256
Optimizer	Adam
Optimizer Betas	(0.9, 0.999)
Optimizer Epsilon	1e-08
Learning Rate Scheduler Type	Linear

Table 2: Configured Training Hyperparameters for Model Development

Final Project Report-Team 6

October 23, 2023

[ ]:
```
PROJECT = "aai520-project" # @param {type:"string"}
GROUP = "aai520-group6" # @param {type:"string"}
MODELS_STR = "distilbert-base-uncased, bert-base-uncased, roberta-base" #␣
 ↪@param {type:"string"}
MODELS = [model.strip() for model in MODELS_STR.split(',')]
TASK = "question-answering" # @param {type:"string"}
DATASET = "squad_v2" # @param {type:"string"}
NUM_TRAIN_EPOCHS = 4  # @param {type:"integer"}
LEARNING_RATE = 2e-5  # @param
GRADIENT_ACCUMULATION_STEPS = 4  # @param {type:"integer"}
PER_DEVICE_TRAIN_BATCH_SIZE = 128  # @param {type:"integer"}
PER_DEVICE_EVAL_BATCH_SIZE = 128  # @param {type:"integer"}
EVALUATION_STRATEGY = 'steps'  # @param {type:"string"}
EVAL_STEPS = 100  # @param {type:"integer"}
SAVE_STRATEGY = "steps"  # @param {type:"string"}
SAVE_STEPS = 100  # @param {type:"integer"}
LOGGING_STEPS = 100  # @param {type:"integer"}
FP16 = True  # @param {type:"boolean"}
DATALOADER_NUM_WORKERS = 2  # @param {type:"integer"}
REPORT_TO = 'tensorboard'  # @param {type:"string"}
LOAD_BEST_MODEL_AT_END = True  # @param {type:"boolean"}
DISABLE_TQDM = False  # @param {type:"boolean"}
PUSH_TO_HUB = True  # @param {type:"boolean"}
OVERWRITE_OUTPUT_DIRECTORY = True  # @param {type:"boolean"}
```

[ ]:
```
#-------------------------------------------------------------------------------#
# CLEAN ENVIRONMENT
#-------------------------------------------------------------------------------#

import gc
gc.collect()

import torch
torch.cuda.empty_cache()
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

#-------------------------------------------------------------------------------#
```

```
# MOUNT GOOGLE DRIVE
#-------------------------------------------------------------------------------#

from pathlib import Path
from google.colab import drive
drive.mount('/content/drive', force_remount=True)
DRIVE_PATH = Path("/content/drive/My Drive/Colab Notebooks")

#-------------------------------------------------------------------------------#
# HANDLE PATHS
#-------------------------------------------------------------------------------#

PROJECT_PATH = DRIVE_PATH / PROJECT
PROJECT_PATH.mkdir(parents=True, exist_ok=True)
CHECKPOINTS_DIR = PROJECT_PATH / "checkpoints"
CHECKPOINTS_DIR.mkdir(parents=True, exist_ok=True)
LOGS_DIR = PROJECT_PATH / "logs"
LOGS_DIR.mkdir(parents=True, exist_ok=True)
VISUALS_DIR = PROJECT_PATH / "visuals"
VISUALS_DIR.mkdir(parents=True, exist_ok=True)

#-------------------------------------------------------------------------------#
# SYMLINK FOR FASTER FILESYSTEM TRAVERSAL
#-------------------------------------------------------------------------------#

SHORTCUT = Path("/content/project")
if not SHORTCUT.exists():
    SHORTCUT.symlink_to(PROJECT_PATH)

#-------------------------------------------------------------------------------#
# INSTALL DEPENDENCIES
#-------------------------------------------------------------------------------#

!pip install \
    -qqq \
    --progress-bar=off \
    datasets \
    evaluate \
    huggingface_hub \
    python-dotenv \
    tensorboardcolab \
    transformers[torch] \
    txtai

#-------------------------------------------------------------------------------#
# IMPORT LIBRARIES & SETUP/CONFIGURE
#-------------------------------------------------------------------------------#
```

```
import json
import os

import evaluate
from datasets import load_dataset
from dotenv import load_dotenv
from evaluate.visualization import radar_plot
from huggingface_hub import EvalResult, ModelCard, ModelCardData, RepoCard,␣
 ↪login
from transformers.trainer_utils import get_last_checkpoint
from txtai.pipeline import HFTrainer



# LOGIN TO HUB
load_dotenv(dotenv_path=PROJECT_PATH / ".env")
login(token=os.getenv('HF_TOKEN'))

# IGNORE TRANSFORMER WARNINGS
os.environ['TRANSFORMERS_NO_ADVISORY_WARNINGS'] = 'true'

#-------------------------------------------------------------------------------#
# LOAD DATASET
#-------------------------------------------------------------------------------#

# DATASET
ds = load_dataset(DATASET)
ds.push_to_hub(repo_id=f"{GROUP}/{DATASET}")

#-------------------------------------------------------------------------------#
# ITERATE THROUGH ALL MODELS
#-------------------------------------------------------------------------------#

combined_results = {}

for pretrained_model in MODELS:

    # RENAME MODEL TO DISTINGUISH FINETUNED MODEL
    finetuned_model = pretrained_model.replace('base', 'finetuned')
    print(f'FINE-TUNING: {finetuned_model}')

    # CONSTRUCT STANDARD IDENTIFER FOR PUBLISHING
    IDENTIFIER = f"{GROUP}/{finetuned_model}-{DATASET}"

    # LOAD CHECKPOINTS
    model_checkpoints = CHECKPOINTS_DIR / finetuned_model
```

```
    model_checkpoints.mkdir(parents=True, exist_ok=True)
    last_checkpoint = get_last_checkpoint(model_checkpoints)

#-------------------------------------------------------------------------------#
# TRAIN MODEL
#-------------------------------------------------------------------------------#

    # use txtai for preconfigured preprocessing of SQuAD datasets
    # https://github.com/neuml/txtai
    trainer = HFTrainer()
    model, tokenizer = trainer(

        # SPECIFICATIONS
        base=last_checkpoint or pretrained_model,
        task=TASK,

        # DATASETS
        train=ds["train"],
        validation=ds["validation"],

        # HYPERPARAMETERS
        num_train_epochs=NUM_TRAIN_EPOCHS,
        learning_rate=LEARNING_RATE,
        gradient_accumulation_steps=GRADIENT_ACCUMULATION_STEPS,
        per_device_train_batch_size=PER_DEVICE_TRAIN_BATCH_SIZE,
        per_device_eval_batch_size=PER_DEVICE_EVAL_BATCH_SIZE,

        # CHECKPOINTS/OUTPUTS
        checkpoint=last_checkpoint,
        output_dir=str(model_checkpoints),

        # STRATEGIES
        evaluation_strategy=EVALUATION_STRATEGY,
        eval_steps=EVAL_STEPS,
        save_strategy=SAVE_STRATEGY,
        save_steps=SAVE_STEPS,
        logging_steps=LOGGING_STEPS,

        # OPTIMIZATIONS
        fp16=FP16,
        dataloader_num_workers=DATALOADER_NUM_WORKERS,

        # SAVING/LOGGING
        report_to=REPORT_TO,
        logging_dir=str(LOGS_DIR),
        load_best_model_at_end=LOAD_BEST_MODEL_AT_END,
        disable_tqdm=DISABLE_TQDM,
```

```
        # PUBLISH TO HUB
        push_to_hub=PUSH_TO_HUB,
        hub_model_id=IDENTIFIER,
        hub_token=os.getenv('HF_TOKEN')

    )

#--------------------------------------------------------------------------------#
# VALIDATE MODEL
#--------------------------------------------------------------------------------#

    print(f'VALIDATING: {finetuned_model}')
    evaluate.logging.set_verbosity_info()
    results = evaluate.evaluator("question-answering").compute(
        model_or_pipeline=IDENTIFIER,
        data=ds["validation"].select(range(2)),
        metric=DATASET,
        squad_v2_format=True,
    )

    for metric_type, metric_value  in results.items():
        metric_name = metric_type.upper() if metric_type == 'f1' else␣
 ↪metric_type.capitalize()
        evaluate.push_to_hub(
            model_id=IDENTIFIER,
            metric_value=metric_value,
            metric_type=metric_type,
            metric_name=metric_name,
            dataset_type=DATASET,
            dataset_name="SQuAD v2",
            dataset_split="validation",
            task_type="question-answering",
            task_name="Question Answering",
            overwrite=True
        )

    # SAVE METRICS
    metrics_file_path = model_checkpoints / 'metrics.json'
    with open(metrics_file_path, 'w') as f:
        json.dump(results, f, indent=4)

    combined_results[pretrained_model] = results

#--------------------------------------------------------------------------------#
# CREATE MODEL CARD FOR HUGGINGFACE
#--------------------------------------------------------------------------------#
```

```
    # ITERATE AND ADD ALL METRICS
    eval_results = [
        EvalResult(
            task_type=TASK,
            dataset_type=DATASET,
            dataset_name='SQuAD v2',
            metric_type=metric_name.replace('_', ' ').title(),
            metric_value=metric_value
        )
        for metric_name, metric_value in results.items()
    ]

    # CREATE MODEL CARD DATA
    model_card_data = ModelCardData(
        language='en',
        license='mit',
        model_name=finetuned_model,
        eval_results=eval_results,

        model_details=f"""
        ## Abstract
        This model, '{finetuned_model}', is a question-answering chatbot␣
↪trained on the SQuAD dataset, demonstrating competency in building␣
↪conversational AI using recent advances in natural language processing. It␣
↪utilizes a BERT model fine-tuned for extractive question answering.

        ## Data Collection and Preprocessing
        The model was trained on the Stanford Question Answering Dataset␣
↪(SQuAD), which contains over 100,000 question-answer pairs based on␣
↪Wikipedia articles. The data preprocessing involved tokenizing context␣
↪paragraphs and questions, truncating sequences to fit BERT's max length, and␣
↪adding special tokens to mark question and paragraph segments.

        ## Model Architecture and Training
        The architecture is based on the BERT transformer model, which was␣
↪pretrained on large unlabeled text corpora. For this project, the BERT base␣
↪model was fine-tuned on SQuAD for extractive question answering, with␣
↪additional output layers for predicting the start and end indices of the␣
↪answer span.

        ## SQuAD 2.0 Dataset
        SQuAD 2.0 combines the existing SQuAD data with over 50,000␣
↪unanswerable questions written adversarially by crowdworkers to look similar␣
↪to answerable ones. This version of the dataset challenges models to not␣
↪only produce answers when possible but also determine when no answer is␣
↪supported by the paragraph and abstain from answering.
```

```
        """,
        intended_use=f"""
        - Answering questions from the {DATASET} dataset.
        - Developing question-answering systems within the scope of the␣
↪{PROJECT}.
        - Research and experimentation in the NLP question-answering domain.
        """,
        limitations_and_bias=f"""
        The model inherits limitations and biases from the '{pretrained_model}'␣
↪model, as it was trained on the same foundational data.
        It may underperform on questions that are ambiguous or too far outside␣
↪the scope of the topics covered in the {DATASET} dataset.
        Additionally, the model may reflect societal biases present in its␣
↪training data.
        """,
        ethical_considerations=f"""
        This model should not be used for making critical decisions without␣
↪human oversight,
        as it can generate incorrect or biased answers, especially for topics␣
↪not covered in the training data.
        Users should also consider the ethical implications of using AI in␣
↪decision-making processes and the potential for perpetuating biases.
        """,
        evaluation=f"""
        The model was evaluated on the {DATASET} dataset using various metrics.␣
↪These metrics, along with their corresponding scores,
        are detailed in the 'eval_results' section. The evaluation process␣
↪ensured a comprehensive assessment of the model's performance
        in question-answering scenarios.
        """,
        training=f"""
        The model was trained over {NUM_TRAIN_EPOCHS} epochs with a learning␣
↪rate of {LEARNING_RATE}, using a batch size of {PER_DEVICE_TRAIN_BATCH_SIZE}.
        The training utilized a cross-entropy loss function and the AdamW␣
↪optimizer, with gradient accumulation over {GRADIENT_ACCUMULATION_STEPS}␣
↪steps.
        """,
        tips_and_tricks=f"""
        For optimal performance, questions should be clear, concise, and␣
↪grammatically correct.
        The model performs best on questions related to topics covered in the␣
↪{DATASET} dataset.
        It is advisable to pre-process text for consistency in encoding and␣
↪punctuation, and to manage expectations for questions on topics outside the␣
↪training data.
        """
```

```
    )

    # CREATE AND SAVE MODEL CARD
    model_card = ModelCard.from_template(model_card_data)
    model_card_path = model_checkpoints / 'README.md'
    model_card.save(model_card_path)

#-------------------------------------------------------------------------------#
# SAVE AND PUBLISH FINAL MODEL COMPONENTS
#-------------------------------------------------------------------------------#

    # MODEL
    model.save_pretrained(model_checkpoints)
    model.push_to_hub(repo_id=IDENTIFIER,repo_type="model")

    # TOKENIZER
    tokenizer.save_pretrained(model_checkpoints)
    tokenizer.push_to_hub(repo_id=IDENTIFIER, repo_type="model")

    # MODEL CARD
    model_card.push_to_hub(IDENTIFIER)

#-------------------------------------------------------------------------------#
# CLEAN UP BEFORE NEXT ITERATION
#-------------------------------------------------------------------------------#

    del model, tokenizer, trainer
    torch.cuda.empty_cache()
    gc.collect()
```