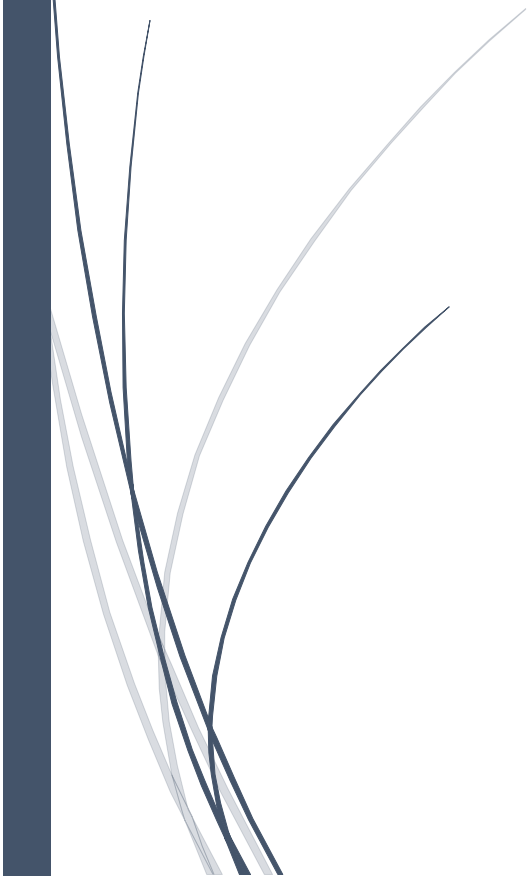
project

BERT-Tuned QA with LoRA



TEAM MEMBERS:

Name	Section
محمد أحمد زكريا الهادى النشار	3
محمد أنور محمد حسنى	3
أحمد رأفت كامل فيشار	1
توماس باسم فوزى	2
محمد أبراهيم عبد الفتاح	3
عبد الرحمن ماجد	2
عبد الله ناصر	3

TABLE OF CONTENTS

PROJECT OVERVIEW :

- This project aims to build an efficient and accurate **Question Answering (QA) system** by **fine-tuning a BERT**-based model using **LoRA** (Low-Rank Adaptation) on the **SQuAD** v2 dataset.
- Illustrative Example of the Final Outcome Targeted by the Project:
  1) **Question "input":**Who were the prominent authors who covered the post-punk era?
  2) **Answer "output":** Jon Savage, Paul Morley, and Ian Penman
  3) **Score:** 0.98 → an indicator of high accuracy.

TRADITIONAL METHODS VS TRANSFORMERS:

- Before the emergence of transformers, let's clarify the old methods used in text processing:
  - **Recurrent Neural Networks** (RNN) and **Long Short-Term Memory** (LSTM) networks are old methods used for processing sequential data such as text or speech. The fundamental idea behind RNNs is that each step in the sequence depends on the previous steps.
  - However, RNNs suffer from a problem known as the "vanishing/exploding gradient problem," which makes it difficult for them to learn long-term dependencies between words in sentences or data.
  - LSTM was developed as a solution to this problem, as it allows for maintaining memory state over long periods, thus making it more efficient than RNNs in learning long-term dependencies in sequences
- Transformer :
  - The Transformer was introduced in 2017 in a research paper titled *"Attention is All You Need"*, marking a paradigm shift in the way sequential data is processed.
  - The key advantage of the Transformer is that, unlike RNN and LSTM models which rely on step-by-step processing, it employs what is known as the **Attention Mechanism**, allowing the model to focus on specific parts of the sequence simultaneously.
  - This mechanism enables the model to compare each word in the sequence with all other words at once, thereby understanding the

relationships between words in a single step. This means the model can "attend" to the entire sequence in parallel.

- Unlike RNNs and LSTMs that process one word at a time, the Transformer operates in **parallel**, processing all tokens in the input simultaneously. This makes it significantly faster and more efficient when handling large-scale data.
- The Transformer architecture is composed of several key components such as **Multi-Head Attention** and **stacked layers**, which allow it to learn more flexible and accurate representations.
- **Key Differences:**
  1. **Sequential vs. Parallel Processing**: Traditional methods process data sequentially (word by word), while the Transformer processes all words at the same time.
  2. **Context Awareness**: The Transformer heavily relies on attention, enabling it to understand context through dynamic relationships between words, regardless of their distance in the sequence.
  3. **Speed and Efficiency**: Due to its parallel nature, the Transformer trains faster and handles larger datasets more efficiently than traditional sequential models.
- The Transformer has revolutionized many applications such as machine translation and large language models (e.g., GPT and BERT), thanks to its ability to model complex relationships in data and deliver accurate results at scale.

THE EVOLUTION FROM TRADITIONAL METHODS TO TRANSFORMERS :

1. Bag-of-Words (BoW) Model

   The **Bag-of-Words** model represents texts as vectors of word frequencies, counting the occurrence of each word in a document without considering word order or grammatical structure. This approach is simple and commonly used in basic text classification tasks.

- **Advantages:** It is easy to implement and computationally efficient. Simply counting word occurrences allows the construction of a document-term matrix, which can be effective in identifying clearly distinct documents.
- **Disadvantages:** The model assumes word independence and ignores contextual relationships and word order. It treats multi-word expressions as separate words and cannot distinguish polysemy (multiple meanings of a word). Additionally, the resulting vectors are typically high-dimensional and sparse, leading to potential issues with overfitting and generalization.

2. TF-IDF (Term Frequency–Inverse Document Frequency)

TF-IDF is an extension of BoW that incorporates the rarity of a word across the entire document corpus. It assigns higher weights to informative words that occur infrequently across documents and downweights common words like articles and prepositions.

- **Advantages:** It improves the relevance of important terms by penalizing high-frequency, low-importance words, making it more effective for document retrieval and classification.
- **Disadvantages:** It remains a shallow statistical model that does not capture word meanings or semantic context. It treats each word as independent and fails to understand nuances or relationships between words. As such, it cannot handle complex language structures or contextual meaning.

3. Word Embeddings (e.g., Word2Vec, GloVe)

Word embeddings such as **Word2Vec** and **GloVe** use deep learning techniques to map each word to a dense vector in a multi-dimensional space, where semantic similarities are captured based on word co-occurrence in large corpora. Words that appear in similar contexts have similar vector representations.

- **Advantages:** These models provide semantically rich representations, enabling detection of word similarities and relationships. Pretrained embeddings can also be reused in various applications, saving training time and computational resources.
- **Disadvantages:** Each word is represented by a single static vector, regardless of context. Thus, words with multiple meanings are represented the same way in all sentences. These models also do not capture syntactic order or handle out-of-vocabulary words effectively.

4. Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM)

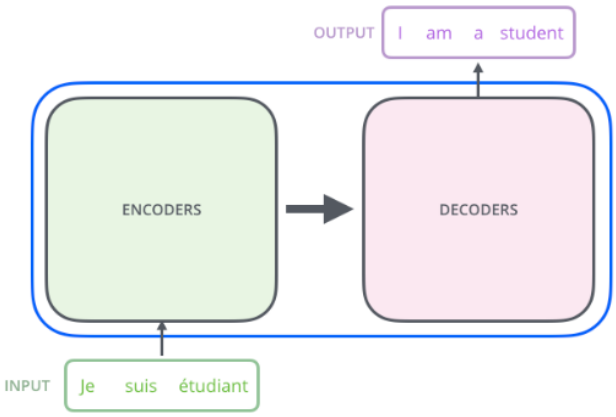
Recurrent Neural Networks (RNNs) are designed to handle sequential data by maintaining a hidden state that evolves over time, making it suitable for modeling temporal dependencies in text.

- **Advantages:** RNNs can process sequences with temporal structure by preserving information from previous steps through the hidden state.
- **Disadvantages:** Traditional RNNs suffer from the **vanishing gradient problem**, where long-range dependencies are difficult to learn. Moreover, RNNs operate sequentially, processing one token at a time, which limits their parallelization and training efficiency.

Long Short-Term Memory (LSTM) networks were introduced as a solution to the vanishing gradient problem. LSTMs include memory cells and gating mechanisms (input, forget, and output gates) to control information flow.

- **Advantages:** LSTMs can retain relevant information over long sequences, mitigating the gradient decay issues. They are better suited to capture long-term dependencies in text.

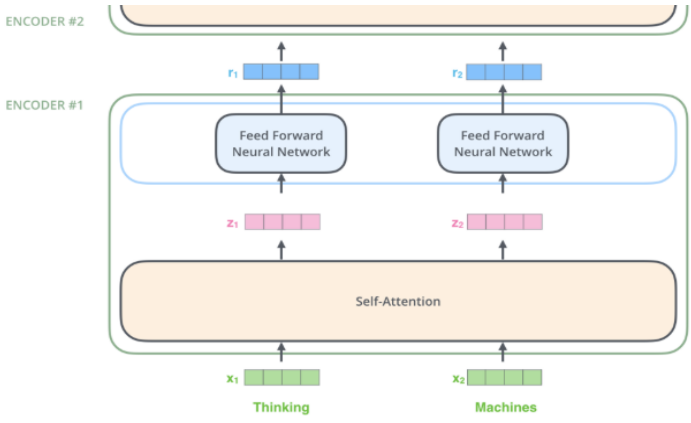
- **Disadvantages:**
Despite improvements over RNNs, LSTMs still rely on sequential processing, limiting computational efficiency and making them slower for long sequences.



5. The Transformer Model

The **Transformer architecture** represents a novel paradigm in natural language processing, first introduced in the seminal 2017 paper *"Attention is All You Need."* It comprises two principal components: the **Encoder** and the **Decoder**, each structured as a stack of uniform layers. Unlike earlier models, the Transformer relies exclusively on **self-attention mechanisms**, entirely dispensing with recurrent structures.

Rather than propagating hidden states sequentially from one word to the next, the attention mechanism enables each word to be processed in parallel by referencing all other words in the input sequence simultaneously. This parallelism allows the model to allocate computational resources more effectively, focusing on the most contextually relevant parts of the text for a given task — thus enhancing both efficiency and performance.



The primary advantage of the Transformer is its **attention mechanism**, which allows the model to automatically focus on important words and account for the relationships between them.

For instance, when the model processes a particular word, the **multi-head attention** layers enable it to link this word with all other words and adapt to the context optimally.

For example, if the sentence includes a reference to time, place, or a shared action, the attention mechanism increases the weight of this information, ensuring that the model can appropriately capture its relevance to the overall meaning.

Unlike RNNs and LSTMs, the Transformer processes all tokens **in parallel**, allowing it to leverage modern parallel hardware (like GPUs) effectively. The **self-attention** mechanism enables the model to weigh the relevance of each token in the sequence relative to all others, regardless of their position. This allows the model to build dynamic, context-aware representations of words.

- **Multi-head Attention:** This component allows the model to capture various types of dependencies and contextual relationships by attending to different parts of the sequence simultaneously.
- **Parallel Processing:** All tokens are processed concurrently, which significantly improves training speed and scalability.
- **Superior Context Modeling:** The Transformer is capable of modeling complex, long-range dependencies more effectively than RNNs or LSTMs.

Thanks to these advantages, the Transformer has become the foundation for modern large language models such as **BERT**, **GPT**, and **T5**, enabling significant breakthroughs in tasks like machine translation, question answering, summarization, and more.

TRADITIONAL METHODS PROBLEMS AND HOW TRANSFORMERS SOLVE THIS

1. SEQUENTIAL PROCESSING

- **Limitation**: In traditional models like RNNs and LSTMs, input sequences were processed sequentially, word by word. This meant the model had to process each token before moving to the next.
- **Impact**: This approach significantly slowed down training and prevented full utilization of parallel processing capabilities on modern hardware like GPUs.
- **Solution with Transformer**: The Transformer leverages parallel processing, allowing all tokens in a sentence to be processed simultaneously. This dramatically improves computational efficiency and speeds up training.

2. VANISHING/EXPLODING GRADIENTS

- **Limitation**: RNNs and LSTMs suffered from vanishing or exploding gradient problems, making it difficult to learn long-term dependencies across extended sequences.
- **Impact**: The model struggled to retain and propagate important information across long distances within a sequence.
- **Solution with Transformer**: By using self-attention mechanisms, the Transformer models direct relationships between all tokens in the sequence, eliminating the need for gradients to propagate through time and thus mitigating this issue.

3. Difficulty Handling Long-Range Dependencies

- **Limitation**: Traditional sequence models found it challenging to capture dependencies between distant words in long sentences.
- **Impact**: These models often focused disproportionately on recent tokens, leading to weak understanding of earlier or distant contextual elements.
- **Solution with Transformer**: The Transformer, particularly through multi-head attention, can attend to all positions in the

input simultaneously, making it highly effective at modeling complex and long-range dependencies.

4. LACK OF PARALLELISM IN TRAINING

- **Limitation**: RNNs and LSTMs required sequential processing, making it impossible to fully leverage parallel computing architectures.
- **Impact**: Training times were longer, which limited the scalability of such models for large-scale applications.
- **Solution with Transformer**: The Transformer architecture processes input in parallel, making full use of modern parallel hardware (e.g., GPUs), thereby significantly accelerating training.

5. NEGLECT OF CONTEXTUAL RELATIONSHIPS

- **Limitation**: Models like Bag of Words (BoW) and TF-IDF ignore word order and contextual relationships.
- **Impact**: This led to shallow representations of meaning, poor handling of ambiguity, and limited performance in tasks like translation or sentiment analysis.
- **Solution with Transformer**: Self-attention allows the Transformer to model the contextual interdependence between all words in a sequence, enabling deeper semantic understanding.

6. COMPUTATIONAL COMPLEXITY AND RESOURCE INTENSITY

- **Limitation**: RNNs and LSTMs required repeated computation across time steps, which was computationally intensive, especially for long sequences.
- **Impact**: This made it difficult to scale to large datasets and increased the overall training time and resource consumption.

- **Solution with Transformer**: The Transformer's multi-head attention mechanism efficiently processes the entire sequence in parallel, reducing complexity and improving scalability.

WHAT IS A TRANSFORMER MODEL?

The transformer model is a type of neural network architecture that excels at processing sequential data, most prominently associated with large language models (LLMs). Transformer models have also achieved elite performance in other fields of artificial intelligence (AI), such as computer vision, speech recognition and time series forecasting.

The transformer architecture was first described in the seminal 2017 paper "Attention is All You Need" by Vaswani and others, which is now considered a watershed moment in deep learning.

WHAT IS SELF-ATTENTION?

The attention mechanism is, in essence, an algorithm designed to determine which parts of a data sequence a model should "pay attention to" at any given moment. The core idea is that, for each word (or token) in a sequence, we don't treat it in isolation but rather take into account the context provided by all the other words.

In your example:

"On Friday, the judge issued a sentence."

when the model tries to interpret the word **"sentence,"** it needs to identify which words in the sentence are most relevant to its meaning. Without attention, a traditional model might struggle to capture these

dependencies effectively, since it typically processes words one by one and could miss how distant words influence each other

HOW SELF-ATTENTION WORKS?

In **self-attention**, every token in the sequence "attends" to every other token to build its next representation. Here's how that happens:

1. **Query, Key, and Value**
   Each token is mapped to three learned vectors: a **query (Q)**, a **key (K)**, and a **value (V)**.
   - The **query** vector represents the token that is "asking" for information.
   - The **key** vector represents the token that could provide that information.
   - The **value** vector is the information itself that gets passed along, weighted by the attention scores.
2. **Computing Attention Scores**
   For each token, we take its query vector and compute dot products with the key vectors of every token in the sequence. These dot products measure how much "attention" the query token should give to each other token.
3. **Softmax Normalization**
   We pass those raw scores through a softmax function so they sum to 1. This gives us a proper probability distribution over which tokens to focus on.
4. **Weighted Sum of Values**
   Finally, each token's new representation is the weighted sum of all the value vectors, where the weights are the normalized attention scores. This lets the model emphasize the most relevant tokens (like **"judge"** and **"issued"** in your example) and de-emphasize less relevant ones.

DIFFERENT TYPES OF TRANSFORMER MODELS CATEGORIZED BASED ON TECHNICAL ARCHITECTURE

1. ENCODER-ONLY TRANSFORMERS

- **Goal**: Text Understanding
- **Examples**:
  - **BERT** (Bidirectional Encoder Representations from Transformers)
  - **RoBERTa**, **DistilBERT**, **ALBERT**
- **Features**:
  - Utilizes only the encoder.
  - Relies on bidirectional self-attention.
  - Ideal for tasks such as text classification, **question answering**, and named entity recognition.
  - Trained using **Masked Language Modeling** technique.

2. DECODER-ONLY TRANSFORMERS

- **Goal**: Text Generation
- **Examples**:
  - **GPT** (Generative Pre-trained Transformer)
  - **GPT-2**, **GPT-3**, **GPT-4**
- **Features**:
  - Uses only the decoder.
  - Relies on causal self-attention (unidirectional).
  - Ideal for text generation, sentence completion, dialogue, and even code generation.
  - Trained with **Autoregressive Language Modeling** technique.

3. ENCODER-DECODER TRANSFORMERS (SEQ2SEQ)

- **Goal**: Sequence-to-Sequence (Transforming one sequence into another)
- **Examples**:
  - **T5** (Text-To-Text Transfer Transformer)
  - **BART**, **MarianMT**, **mT5**
- **Features**:
  - Contains both Encoder and Decoder.
  - Suitable for tasks that involve understanding and generating text simultaneously.
  - Ideal for machine translation, summarization, paraphrasing, and question answering.
  - Trained using techniques like **Denoising** and **Seq2Seq Learning**.

4. VISION TRANSFORMERS (VIT)

- **Goal**: Image Processing using Transformers
- **Examples**:
  - **ViT**, **Swin Transformer**, **DETR**
- **Features**:
  - Divides the image into patches and processes them like a sequence of tokens.
  - Uses the Transformer architecture to understand image content.
  - Suitable for image classification, object detection, and image segmentation tasks.

5. MULTIMODAL TRANSFORMERS

- **Goal**: Processing Multimodal Data (Text + Image + Audio)
- **Examples**:
  - **CLIP**, **BLIP**, **Flamingo**, **GIT**
- **Features**:

- Integrates text and images (or other modalities) within a single model.
- Ideal for tasks like image captioning, visual question answering, and multimodal understanding.

6. RETRIEVAL-AUGMENTED TRANSFORMERS

- **Goal**: Incorporating external knowledge during inference or understanding
- **Examples**:
  - **RETRO** (Retrieval-Enhanced Transformer)
  - **RAG** (Retrieval-Augmented Generation)
- **Features**:
  - Retrieves information from external databases or sources.
  - Combines the retrieved information with the generated output to enhance accuracy.

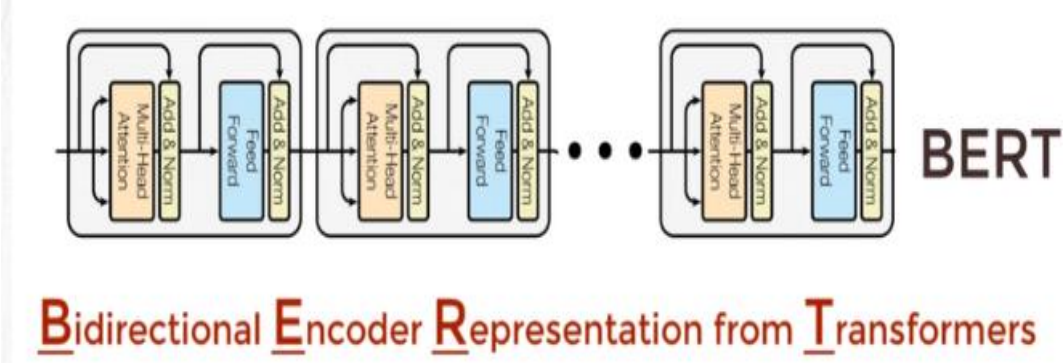
7. COMPACT & EFFICIENT TRANSFORMERS

- **Goal**: Reducing model size and increasing speed while maintaining performance
- **Examples**:
  - **DistilBERT**, **TinyBERT**, **MobileBERT**, **ALBERT**
- **Features**:
  - Smaller, faster models designed for resource-constrained environments.
  - Ideal for mobile and embedded applications.

BERT:

- **Bidirectional Context:**
  Unlike earlier models that read text left-to-right or right-to-left, BERT looks at both sides of each token simultaneously via its self-attention mechanism.

- **Encoder-Only Architecture:**
  It uses only the transformer's encoder layers (no decoder layers like in translation models). The "Base" version has 12 layers; the "Large" version has 24.
- **Pre-training Tasks:**
  Trained on massive text corpora (e.g., Wikipedia) using two objectives:
  1. **Masked Language Modeling (MLM):**
     Randomly mask about 15% of tokens ([MASK]) and train the model to predict them using bidirectional context.
  2. **Next Sentence Prediction (NSP):**
     Train to predict whether sentence B follows sentence A in the original text, improving understanding of inter-sentence relationships.



HOW BERT WORKS ?

1. **Input Representation:**
   - Special tokens: [CLS] at the start; [SEP] at the end of each sentence.
   - Each token embedding is the sum of:
     - **Token Embedding** (word-piece)
     - **Position Embedding**
     - **Segment Embedding** (when there are multiple sentences)
2. **Multi-Head Self-Attention:**
   Each position attends to all positions in the sequence, learning

   different "views" of Context.

3. **Feed-Forward Networks:**
   After attention, there are fully connected layers with ReLU activations.

4. **Layer Normalization & Residual Connections:**
   Used in each sub-layer to stabilize training and preserve gradient flow.

USES OF BERT:

After pre-training, BERT can be **fine-tuned** on downstream tasks with minimal additional layers:

1. **Question Answering:**
   e.g., SQuAD—add two output layers to predict the start and end positions of an answer span.
2. **Text Classification:**
   Binary or multi-class tasks (sentiment analysis, topic classification) using the [CLS] token's representation.
3. **Named Entity Recognition (NER):**
   Label each token (or subword) with entity categories like Person, Organization, Location.
4. **Sentence Pair Classification:**
   Tasks like semantic similarity or paraphrase detection, using both sentences as input.
5. **Cloze & Masked Generation:**
   Fill-in-the-blank style tasks or limited masked generation.

LORA :

- **LoRA is a technique used to fine-tune large pre-trained models**
  - In a typical neural network, weight matrices W are large.
  - LoRA decomposes the change in W into two smaller matrices:
  - $\Delta W \approx A \times B$(where A and B are low-rank)
  - These matrices A and B are the only ones trained.

    - The original weight W remains unchanged.
- Efficient fine-tuning: Only a few extra parameters are trained.
- Memory-saving: Great for limited hardware (e.g., laptops or edge devices).
- Fast adaptation: Enables quick training on new tasks.
- Maintains original model: The base model stays frozen, so it can be reused

DATASET: "SQUAD V2"

The **Stanford Question Answering Dataset, Version 2.0 (SQuAD v2)** is an extension of the original SQuAD dataset, widely used for training and evaluating text-based question-answering (QA) models. Its key characteristics are:

1. **Source and Content**
    - Drawn from English-language Wikipedia articles.
    - Contains around **150,000 paragraph–question pairs**, of which roughly **50,000 questions are unanswerable** based on the paragraph.
2. **Answerable vs. Unanswerable Questions**
    - In SQuAD v1, every question had an answer span in the paragraph.
    - In v2, unanswerable questions are added, forcing models to first **decide** if a question can be answered before attempting to extract a span.
    - This makes the task more realistic, since real-world users often ask questions that the provided text doesn't cover.
3. **Data Structure**
    - **Paragraphs**: Short passages (typically 2–5 sentences) each accompanied by multiple questions.
    - **Questions**: Each question entry includes fields such as:
        - is_impossible: a Boolean indicating whether the question has no answer in the paragraph.
        - answers: a list of answer objects; for answerable questions, each object gives the exact character span(s) in the paragraph.
    - Stored in a JSON format that's easy to load with libraries like Hugging Face Datasets.

4. **Evaluation Objectives & Metrics**
    1. **Answerability Classification**: Predict the is_impossible flag (binary classification).
    2. **Span Extraction**: For answerable questions, predict the start and end token indices of the answer.
    3. **Primary Metrics**:
        - **Exact Match (EM)**: Percentage of predictions that match any ground-truth answer exactly.
        - **F1 Score**: Overlap between predicted and ground-truth answer tokens (token-level precision & recall).

5. **Why Use SQuAD v2?**
    - Provides a realistic "no-answer" challenge, teaching models to **abstain** rather than hallucinate incorrect answers.
    - Better simulates real-world QA scenarios in which not every question is covered by the text.
    - Serves as a rigorous benchmark that drives research into model calibration and reducing hallucinations.

HYPERPARAMETERS:

- Batch size: 16
- Epochs: 3
- Learning rate: default (with AdamW)
- Logging: every 100 steps
- Eval strategy: every epoch
- Use of Trainer API for training

TRAINING STRATEGY:

- Used Trainer class from Hugging Face.
- LoRA-wrapped model passed into trainer.
- wandb used for tracking and logging.
- Validation and checkpointing configured.

CODE AND OUTPUT RESULT:

Code

```
from transformers import AutoTokenizer, AutoModelForQuestionAnswering, pipeline
from peft import PeftModel, PeftConfig

config = PeftConfig.from_pretrained("MohamedShakhsak/bert-qa-squad2_V2")

base_model = AutoModelForQuestionAnswering.from_pretrained(config.base_model_name_or_path)
lora_model = PeftModel.from_pretrained(base_model, "MohamedShakhsak/bert-qa-squad2_V2")

merged_model = lora_model.merge_and_unload()

tokenizer = AutoTokenizer.from_pretrained(config.base_model_name_or_path)

qa_pipeline = pipeline("question-answering", model=merged_model, tokenizer=tokenizer)
```

```
examples = [
    {
        "question": "What is the capital of France?",
        "context": "Paris is the capital and most populous city of France."
    },
    {
        "question": "When was the iPhone first released?",
        "context": "The first iPhone was released by Apple Inc. on June 29, 2007."
    }
]

for example in examples:
    answer = qa_pipeline(example)
    print(f"Q: {example['question']}\nA: {answer}\n")
```

Result

```
Q: What is the capital of France?
A: {'score': 0.010731427930295467, 'start': 0, 'end': 5, 'answer': 'Paris'}

Q: When was the iPhone first released?
A: {'score': 0.2492296759414673, 'start': 47, 'end': 60, 'answer': 'June 29, 2007'}
```