THE GEORGE WASHINGTON UNIVERSITY

WASHINGTON, DC

Quora Question Pair Similarity Classification

Group 5 Final Project Report

Members:
Abhradeep Das (G38747350)
Gourab Mukherjee (G32241729)
Richik Ghosh (G31267506)
Shreya Sahay (G36642286)

INTRODUCTION

Quora is a dynamic platform that facilitates the acquisition and dissemination of knowledge across a vast spectrum of subjects. Each month, it attracts over 100 million visitors who actively engage by posing questions and interacting with experts who provide insightful answers. This high level of engagement underscores Quora's role as a significant hub for information exchange. However, the platform's immense popularity also presents challenges, notably the proliferation of similarly phrased questions. This redundancy forces users to invest considerable time sifting through numerous entries to locate the most pertinent responses, while contributors often find themselves addressing repetitive queries instead of offering unique perspectives.

To address these issues, this project focuses on developing a robust system to classify whether pairs of questions are contextually similar. Utilizing the Quora Question Pairs dataset, the project employs a combination of classical machine learning models, neural network-based approaches, and state-of-the-art transformer-based models. By leveraging advanced natural language processing (NLP) techniques and sophisticated architectural frameworks, the goal is to accurately identify duplicate or near-duplicate questions. This classification system aims to enhance the user experience by streamlining the knowledge-sharing process, reducing redundancy, and ensuring that both seekers and contributors can engage more efficiently and effectively on the platform.

Through the implementation of these methodologies, the project aspires to not only mitigate the issue of duplicate questions but also to contribute to the broader field of NLP by demonstrating the efficacy of various modeling approaches in understanding and interpreting human language nuances. Ultimately, this endeavor seeks to optimize Quora's infrastructure, fostering a more organized and user-friendly environment that maximizes the platform's potential as a premier destination for knowledge exchange.

SHARED WORKFLOW

Shared Work Outline:

1. Exploratory Data Analysis (EDA)

EDA: Conducted exploratory data analysis to understand the distribution of data, identify any patterns, and detect anomalies. This step included visualizations and statistical summaries.

2. Data Preprocessing

- **Data Normalization:** Converting text to lowercase, expanding contractions, and standardizing symbols (e.g., currency, percentages).
- **Data Cleaning:** Removing non-word characters and HTML tags, and processing large numbers into a readable format.
- **Stemming:** Applied the PorterStemmer to reduce words to their root forms.

3. Feature Extraction

- **Tokenization:** Tokenizing the questions using a word tokenizer.
- **Feature Categories:** Extracted 37 features that fall into three main categories:
- **Fuzzy Features:** Including fuzz_ratio, fuzz_partial_ratio, etc., to measure word-to-word fuzzy similarity.
- **Token Features:** Analysis of stopwords and non-stopwords (e.g., common non-stopwords, common stopwords, word size difference).
- **Common Subsequence Features:** Measuring similarity between parts of sentences (e.g., largest_common_subsequence, jaccard_similarity).

4. Vectorization

- Weighted TF-IDF: Implemented weighted TF-IDF scores, leveraging the spaCy model for word embeddings. Combined the generated TF-IDF scores with word embeddings to enhance the feature representation.
- BERT Embeddings: Created BERT embeddings

5. Model Training and Comparison

- Classical Models with TF-IDF and BERT embeddings: Trained classical machine learning models (e.g. Logistic Regression, Random Forest, Naïve Bayes) using the weighted TF-IDF and the BERT embedding features.
- DL Models: Trained and evaluated DL models: LSTM and GRU
- Transformer Models: Applied different variations of the BERT architecture

6. Streamlit app

7. Final Model and Evaluation

- Compared and evaluated all the applied models
- Evaluation Metrics: Used the F1 macro score for model evaluation.

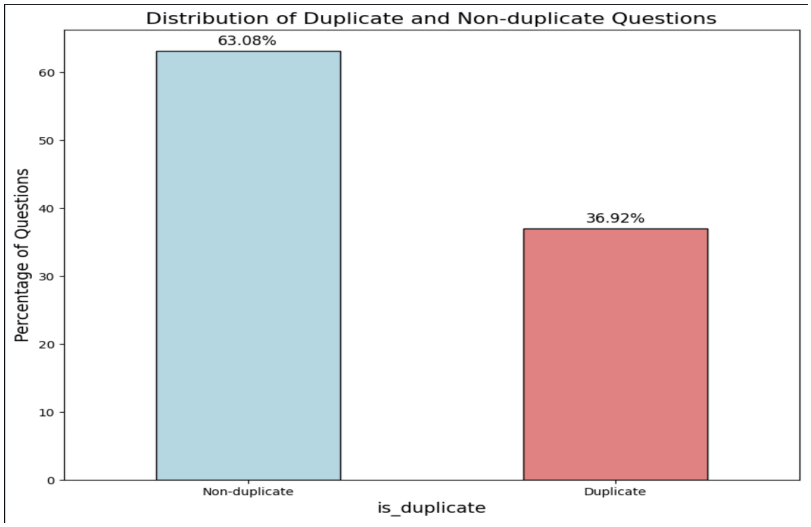
DESCRIPTION OF THE DATASET

The dataset for this project is sourced from a Kaggle competition (link) which had the same goal as our project focused on Quora question pairs. It comprises approximately 404,000 rows of question pairs and includes six key features:

- id: the identifier of a training set question pair.
- qid1, qid2: unique identifiers of each question.
- question1, question2: the full text of each question.
- is_duplicate: the target variable, which is set to 1 if the questions have essentially the same meaning and 0 otherwise.

13	27	28	What was your first sexual experience like?	What was your first sexual experience?	1
14	29	30	What are the laws to change your status from a student visa to a green card in the US, how do they compare to the immigration laws in Canada?	What are the laws to change your status from a student visa to a green card in the US? How do they compare to the immigration laws in Japan?	0
15	31	32	What would a Trump presidency mean for current international master's students on an F1 visa?	How will a Trump presidency affect the students presently in US or planning to study in US?	1
16	33	34	What does manipulation mean?	What does manipulation means?	1
17	35	36	Why do girls want to be friends with the guy they reject?	How do guys feel after rejecting a girl?	0
18	37	38	Why are so many Quora users posting questions that are readily answered on Google?	Why do people ask Quora questions which can be answered easily by Google?	1
19	39	40	Which is the best digital marketing institution in banglore?	Which is the best digital marketing institute in Pune?	0
20	41	42	Why do rockets look white?	Why are rockets and boosters painted white?	1
21	43	44	What's causing someone to be jealous?	What can I do to avoid being jealous of someone?	0
22	45	46	What are the questions should not ask on Quora?	Which question should I ask on Quora?	0
23	47	48	How much is 30 kV in HP?	Where can I find a conversion chart for CC to horsepower?	0
24	49	50	What does it mean that every time I look at the clock the numbers are the same?	How many times a day do a clock's hands overlap?	0
25	51	52	What are some tips on making it through the job interview process at Medicines?	What are some tips on making it through the job interview process at Foundation Medicine?	0
26	53	54	What is web application?	What is the web application framework?	0
27	55	56	Does society place too much importance on sports?	How do sports contribute to the society?	0
28	57	58	What is best way to make money online?	What is best way to ask for money online?	0
29	59	60	How should I prepare for CA final law?	How one should know that he/she completely prepare for CA final exam?	1
30	61	62	What's one thing you would like to do better?	What's one thing you do despite knowing better?	0
31	63	64	What are some special cares for someone with a nose that gets stuffy during the night?	How can I keep my nose from getting stuffy at night?	1
32	65	66	What Game of Thrones villain would be the most likely to give you mercy?	What Game of Thrones villain would you most like to be at the mercy of?	1
33	67	68	Does the United States government still blacklist (employment, etc.) some United States citizens because their political views?	How is the average speed of gas molecules determined?	0

The dataset exhibits class imbalance, with non-duplicate question pairs (63.08%) significantly outnumbering duplicate pairs (36.92%).



METHODOLOGY

1. **Data Preprocessing**

The first step was pre-processing of the data, which included the following steps:

- **Data Normalization**: Transformed text into lowercase to ensure uniformity across the dataset. Expanded common contractions (e.g., "don't" to "do not") to enhance clarity and consistency. Standardized symbols by converting currency signs (e.g., $ to "dollars") and percentages (e.g., 50% to "fifty percent") into their textual equivalents.
- **Data Cleaning**: Removed non-word characters such as special symbols and punctuation marks that did not contribute to the textual meaning. Stripped HTML tags to clean up any embedded markup code within the data. Reformatted large numerical values, converting them into a more interpretable format (e.g., "1,000,000" to "one million").
- **Stemming**: Applied the PorterStemmer algorithm to reduce words to their root forms, minimizing vocabulary size while retaining semantic essence (e.g., "running" and "runner" both mapped to "run"). This step helped to improve model generalization during training.

2. Feature Extraction

Tokenization**:** We began by tokenizing the questions using a word tokenizer. This step is crucial as it breaks down the text into individual words or tokens, facilitating further analysis and feature extraction.

Feature Categories: We extracted a total of 37 features, categorized into three main groups: Fuzzy Features, Token Features, and Common Subsequence Features. Each category provides a different perspective on the textual similarity, ensuring a comprehensive analysis.

1. **Fuzzy Features:**

    - **Fuzz Ratio:** This feature measures the overall similarity between two strings. It computes a ratio by comparing the characters in the strings, providing a score between 0 and 100, where 100 indicates a perfect match.
    - **Fuzz Partial Ratio:** Unlike the fuzz ratio, the partial ratio compares substrings of the given strings, making it useful for identifying partial matches or when one string is a substring of another.
    - **Fuzz Token Sort Ratio:** This feature tokenizes the strings, sorts the tokens, and then calculates the fuzz ratio. This helps in identifying matches where the order of words differs but the content is similar.

    - **Fuzz Token Set Ratio:** This extends the token sort ratio by considering the union and intersection of token sets, thus capturing similarity even when words are repeated or extra words are present.
2. **Token Features:**
    - **Common Non-Stopwords:** This feature counts the number of common non-stopwords between two strings. Non-stopwords are the essential words that carry the primary meaning of the sentences, excluding common stopwords like "and," "the," etc.
    - **Common Stopwords:** Conversely, this feature counts the common stopwords. Although stopwords are generally filtered out, their presence can sometimes be indicative of structural similarity.
    - **Word Size Difference:** This calculates the difference in the number of words between the two strings. It helps in understanding the length disparity, which might affect similarity scores.
    - **Stopword Ratio:** This feature computes the ratio of stopwords to the total number of words in a string. It helps in understanding the structural composition of the text.
3. **Common Subsequence Features:**
    - **Largest Common Subsequence (LCS):** This feature finds the length of the longest subsequence present in both strings.
    - **Jaccard Similarity:** This measures the similarity between finite sample sets, defined as the size of the intersection divided by the size of the union of the sample sets. For text, it compares the set of words or n-grams between the strings.
    - **Longest Common Prefix:** This feature identifies the longest initial segment that is common in both strings, useful for detecting similarity in the beginning parts of the text.

3. Vectorization

To transform textual data into numerical representations suitable for machine learning models, advanced vectorization techniques were implemented to capture both the semantic meaning and contextual relationships within the text.

- **Weighted TF-IDF:** Calculated Term Frequency-Inverse Document Frequency (TF-IDF) scores to assign importance weights to words based on their frequency and rarity within the dataset. Leveraged the spaCy model to generate pre-trained word embeddings, which provided a richer representation of words in high-dimensional space. Combined the TF-IDF weights with the word embeddings, effectively merging frequency-based statistical importance with semantic context to create enhanced feature vectors.

- **BERT Embeddings:** Utilized BERT (Bidirectional Encoder Representations from Transformers) to generate contextualized word embeddings that captured the nuanced relationships between words within their specific contexts. These embeddings provided a deeper understanding of sentence-level meaning, significantly improving downstream model performance by encoding both syntactic and semantic information.

MODELING

The handcrafted features were combined with 300-dimensional word embeddings generated using TF-IDF weighted vectors and SpaCy's `en_core_web_lg` model. This approach captured both statistical importance and semantic relationships. The 37 handcrafted features, combined with the embeddings, resulted in a comprehensive 637-feature matrix, representing each question pair. This enriched feature set served as input for the machine learning models, enabling robust performance in question similarity detection.

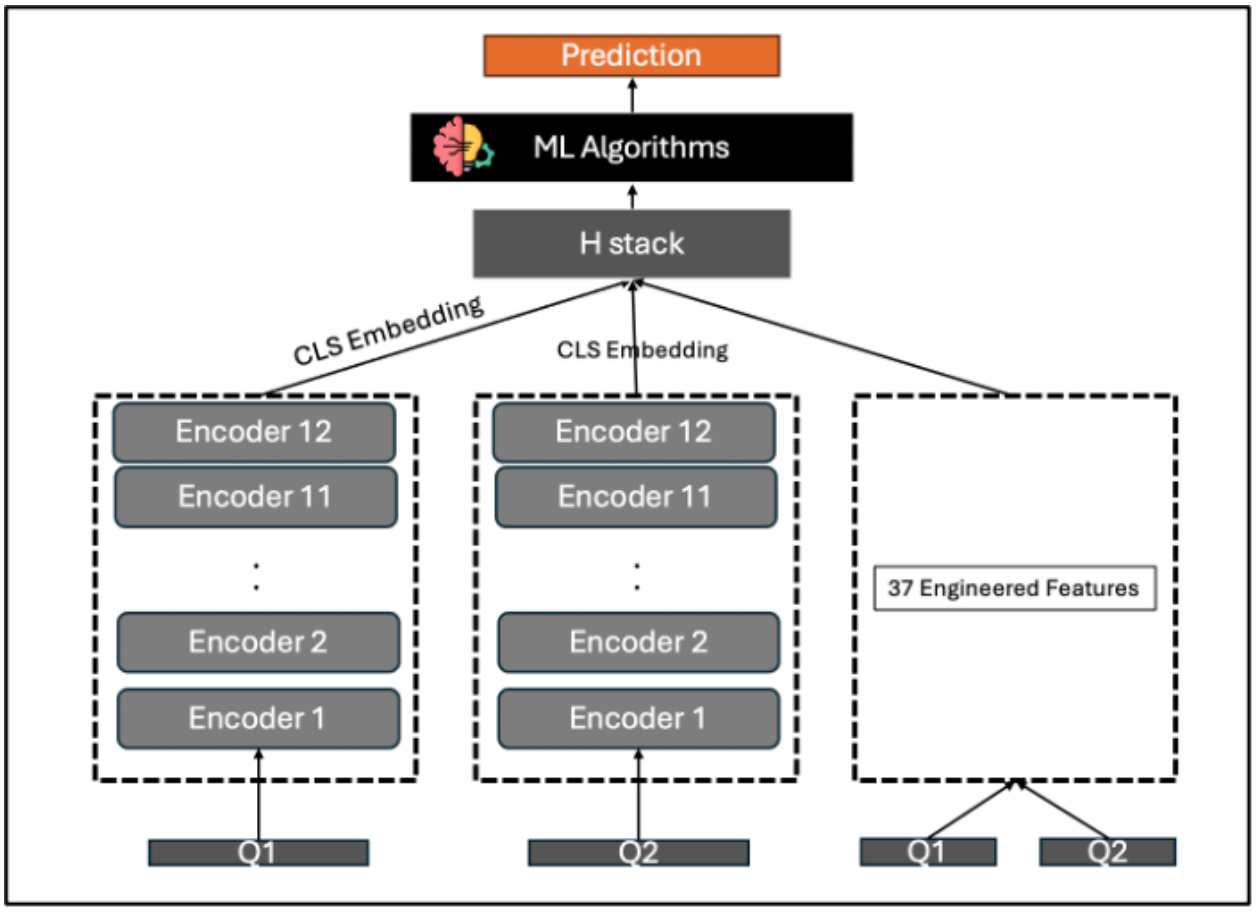
- Used Logistic Regression as a simple and efficient baseline model.
- Used Gaussian Naïve Bayes with standard scaling for probabilistic modeling of normally distributed features.
- Used XGBoost, a powerful gradient boosting algorithm, was used with hyperparameter tuning via RandomizedSearchCV. Key parameters included `n_estimators` (50, 100, 150), `learning_rate` (0.01, 0.1, 0.2), `max_depth` (3, 5, 7), `subsample` (0.7, 0.9), `colsample_bytree` (0.5, 0.7, 0.9, 1.0), and `gamma` (0, 0.1, 0.2). The tuning involved 5 random combinations across 3-fold cross-validation to optimize model performance and generalization.
- Used Random Forest, an ensemble method combining multiple decision trees, was tuned using key parameters: `n_estimators` (50–300) to balance accuracy and computation, `max_depth` (10–None) to capture complexity, `min_samples_split` (2–10) and `min_samples_leaf` (1–4) to prevent overfitting, and `bootstrap` (True/False) for randomness. This tuning optimized model performance and generalization.

GENERATION OF BERT EMBEDDINGS

- We generated BERT-based CLS embeddings for the question1 and question2 features to capture contextual semantics for comparing question pair similarities.
- We fetched raw question pairs from the training data and stored them in a custom dataset class, QuestionsDataset(), for efficient batch processing.
- We designed a compute_batch_embeddings() function to generate embeddings:
    - We used BertTokenizer (bert-base-uncased) to tokenize questions and convert them into token IDs.
    - We extracted CLS token embeddings from outputs.last_hidden_state[:, 0, :].
- We processed questions in batches of 8 using a DataLoader, resulting in (8, 768)-dimensional embeddings for each batch.

- We obtained an (N, 768) embedding matrix for 404,290 question pairs, where N is the total number of samples.
- We combined the generated embeddings with 37 previously created features into a single feature matrix.
- Finally, we saved the feature matrix in .pt format for downstream modeling, including columns: {id, q1_feats_bert, q2_feats_bert, features, labels}.

CLASSICAL MODELS USING BERT EMBEDDINGS



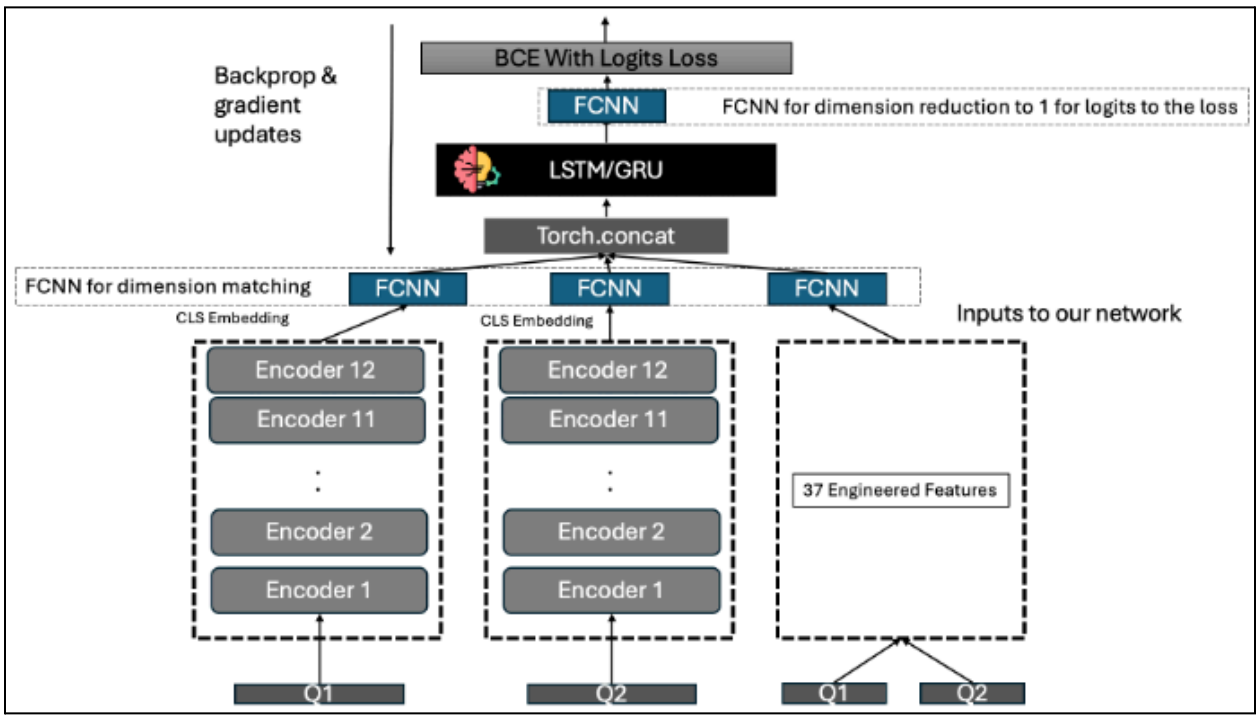
ML models architecture

We designed a dynamic machine learning pipeline tailored to specific algorithms, incorporating appropriate preprocessing steps and hyperparameters for optimal performance.

Algorithms and Configurations:

- **Random Forest**:
    - Selected for handling high-dimensional data and complex relationships.
    - Configured with n_estimators=100, max_depth=None, min_samples_split=2, and min_samples_leaf=1 to capture detailed patterns and encourage overfitting in base estimators for improved ensemble performance.
- **Logistic Regression**:
    - Chosen as a baseline for its simplicity and efficiency on linearly separable data.
    - Used StandardScaler to standardize features and set max_iter=1000 to ensure convergence on complex datasets.
- **XGBoost**:
    - Included for its robust performance on structured and imbalanced datasets.
    - Configured with use_label_encoder=False for compatibility and eval_metric='logloss' for classification accuracy. No standardization was required due to its scaling robustness.
- **Naïve Bayes**:
    - Gaussian Naïve Bayes was selected for its simplicity and probabilistic modeling of normally distributed features.
    - Applied StandardScaler to address numerical instability, achieving a fast and interpretable benchmark.
- **Support Vector Machine (SVM)**:
    - Tested with an RBF kernel for non-linear relationships but encountered inefficiency, causing runtime issues and crashing the IDE.

DEEP LEARNING MODELS USING BERT EMBEDDINGS



Deep Learning Models Architecture

We implemented GRU and LSTM networks, processing data using a DataLoader with a batch size of 32 for efficient loading.

Pipeline Design:

- We defined a CustomDataset() to structure each sample, including the feature matrix.
- We designed two models to process embeddings of question1 (q1) and question2 (q2), along with additional features:
    - We passed q1 and q2 embeddings (size 768) through a shared fully connected layer, reducing their dimensions to hidden_dim=256.
    - We processed additional features with a separate fully connected layer, transforming them into the same 256-dimensional space for consistent representation.
- We concatenated the processed embeddings of q1, q2, and additional features into a single tensor of size [batch_size, 3 * hidden_dim] as input to the model's head (LSTM or GRU).

Head Processing:

- For both LSTM and GRU heads:
    - We reshaped the input to [batch_size, 1, 3 * hidden_dim] and passed it through the respective head.

- We extracted the last hidden state (head_out[-1]) of size [batch_size, hidden_dim] as the final output.

Training Process:

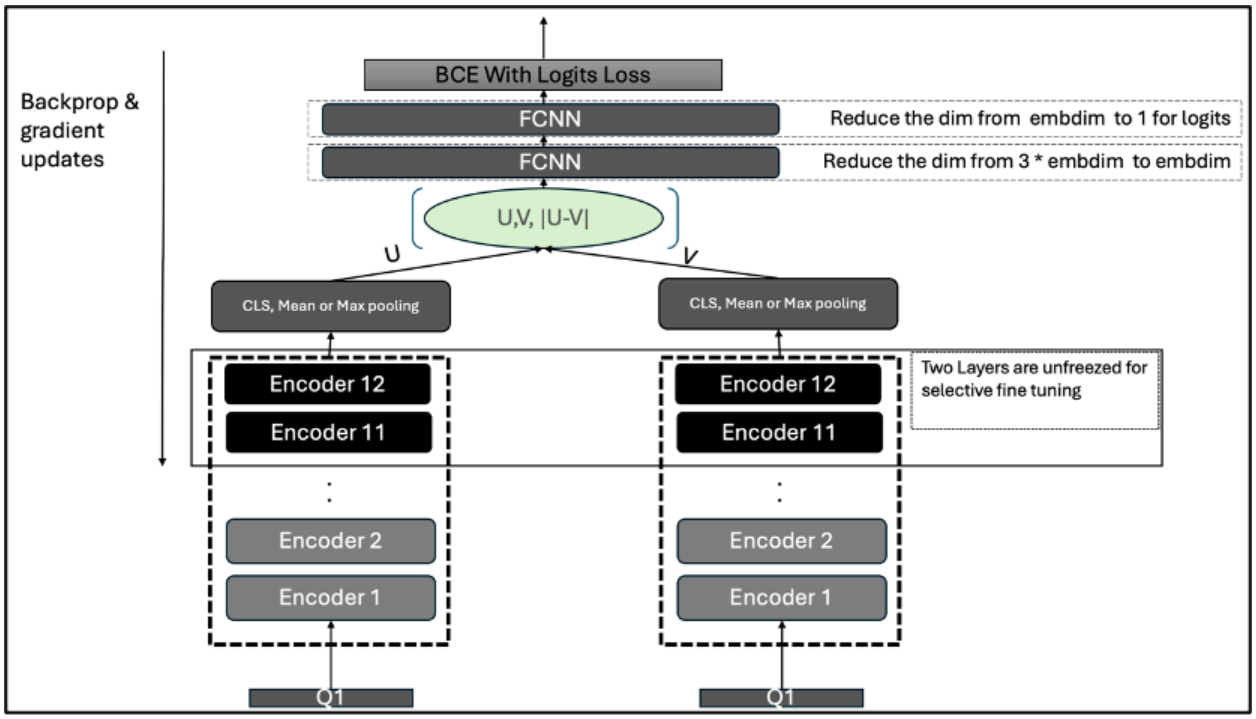
- We used checkpointing to save the model's state whenever a new best validation loss was achieved, enabling us to resume or evaluate the model later.
- We passed the head's output through a fully connected layer to produce a single prediction.

Challenges and Results:

- We experimented with learning rates, epochs, and batch sizes to improve performance, but the loss oscillated between 0.65 and 0.69 for both models.
- We achieved a final loss of 0.65 for LSTM and 0.68 for GRU, suggesting the architecture required further adjustments to improve performance.

TRANSFORMER BASED MODELLING USING BERT EMBEDDINGS

Transformers: We have used 5 different types of architectures of transformers to get the classification done.



Siamese BERT model architecture

- We implemented a modified Siamese Sentence BERT (SBERT) framework to generate semantically rich embeddings and incorporated fine-tuning to adapt it to the question similarity task.
- SBERT, particularly suited for capturing nuanced relationships between text pairs, was initialized with pre-trained weights from sentence-transformers/paraphrase-MiniLM-L6-v2.

Model Architecture:

- We used a Siamese design with two identical SBERT encoders processing question1 (q1) and question2 (q2) independently.
- The last two layers of the SBERT encoder were selectively unfrozen for fine-tuning, enabling task-specific adaptation while avoiding overfitting by keeping earlier layers frozen.
- For each question, the SBERT encoder produced contextual embeddings using three pooling strategies: mean pooling, max pooling, and CLS token pooling. We found mean pooling to perform best.
- The embeddings of q1 (u) and q2 (v) were combined into a concatenated vector comprising three components: u, v, and the absolute difference |u - v|. Including |u - v| improved results by explicitly highlighting semantic dissimilarity.
- The concatenated vector (dimensionality 3 * embedding_dim = 1154) was passed through two fully connected layers (FCNNs):
    - The first FCNN reduced the dimensionality to embedding_dim (384) using ReLU activation and dropout for regularization.
    - The second FCNN further reduced the output to a single logit representing the predicted similarity.

Data Processing:

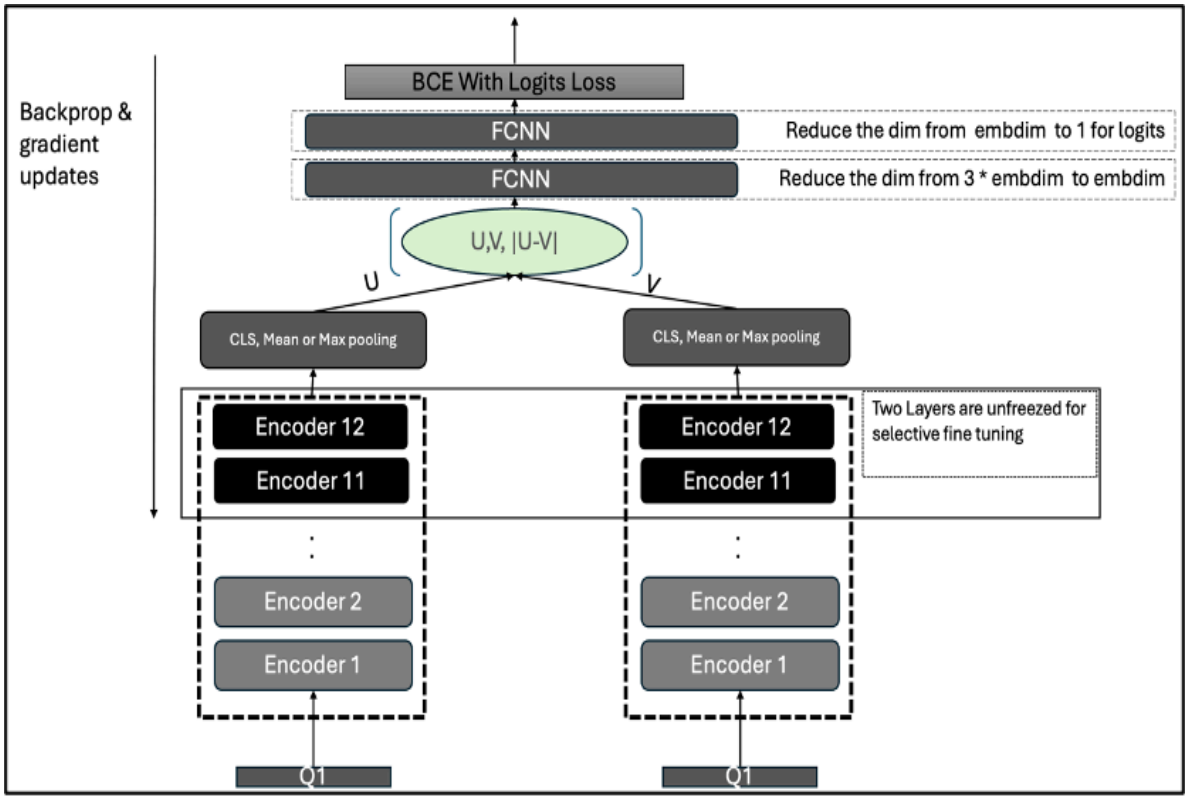
- We created a SiameseQuestionsDataset class to preprocess question pairs (q1 and q2).
    - Each question was tokenized separately with padding, truncation, and special tokens, ensuring compatibility with the SBERT tokenizer.
    - The dataset returned tokenized inputs (input_ids, attention_mask) and labels as PyTorch tensors, structured for efficient batching and training through PyTorch's DataLoader.

Training Process:

- We trained the model for 10 epochs with a batch size of 32, balancing computational efficiency and sufficient updates per epoch.
- We implemented checkpointing to save the model's state whenever a new best validation loss was achieved, allowing us to resume training or evaluate the best-performing model.

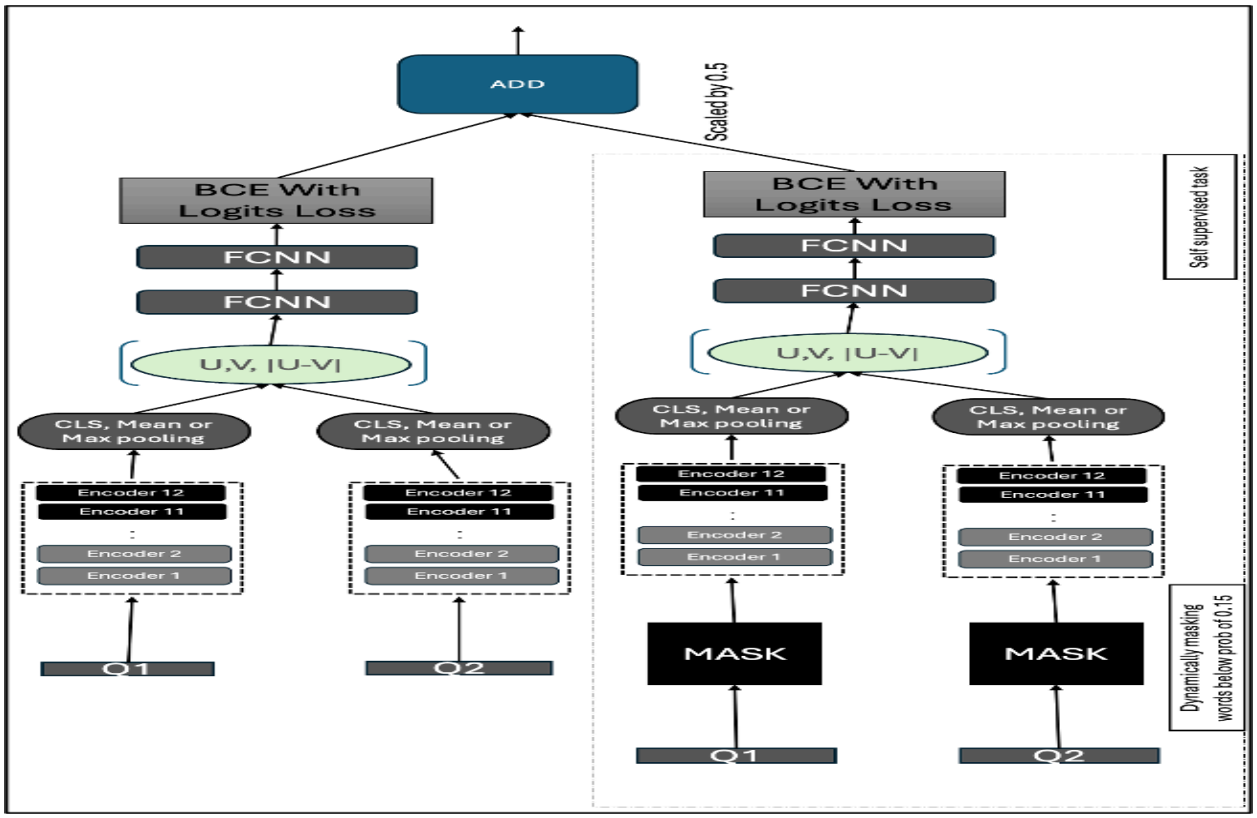
- Fine-tuning was performed by selectively unfreezing the last two layers of the SBERT encoder, keeping the rest frozen.
    - Gradients were computed and updated only for the unfrozen layers, enabling task-specific learning while preserving the pre-trained semantic representations in frozen layers.

1. Siamese SBERT (Bi-Encoder) with FCNN:



- **Base Model**: Leveraged a pre-trained SBERT model (paraphrase-MiniLM-L6-v2), which is a lighter variant of BERT optimized for sentence similarity tasks.
- **Selective Fine-Tuning**: Only the last two encoder layers were unfrozen, allowing the model's top layers to adapt to the question similarity task while retaining robust, pre-trained semantic representations in the frozen layers below.
- **Pooling Strategies**: Compared mean pooling, max pooling, and CLS token pooling for producing fixed-length embeddings. Mean pooling consistently yielded better semantic representations and improved final performance metrics.
- **Embedding Combination**: Incorporated (u, v, |u - v|) into the final representation, ensuring the model could leverage both shared and contrasting features between question embeddings.
- **Outcome**: Achieved strong baseline results. The selective fine-tuning and mean pooling synergy offered a solid balance between computational efficiency, stability, and accuracy.

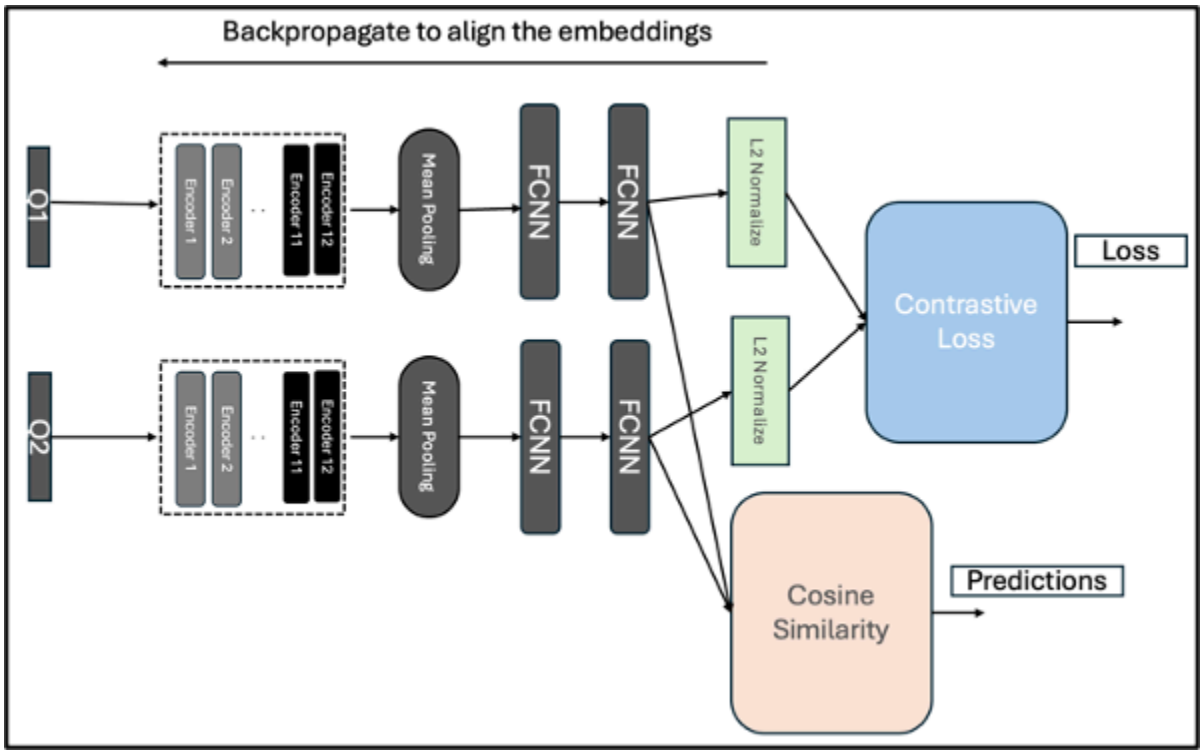
2. Siamese SBERT with Dynamic Masking (Bi-Encoder)



:

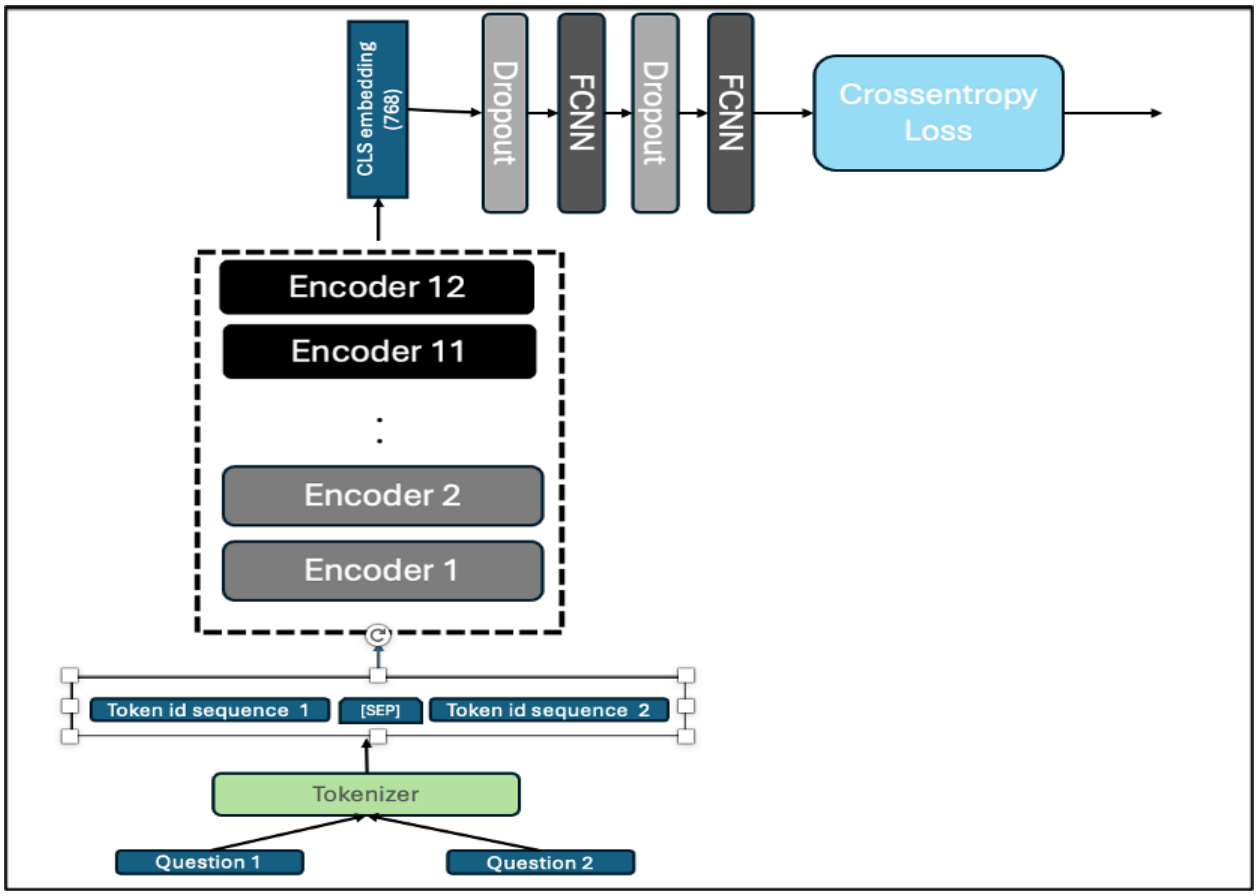
- **Base Model**: Began with the same SBERT backbone and initial setup as above.
- **Selective Fine-Tuning**: Again, only the top two SBERT encoder layers were trainable, ensuring that the network's pretrained semantic core remained intact.
- **Auxiliary Task (Dynamic Masking)**: Introduced a secondary self-supervised objective by randomly masking tokens in the input (except special tokens) during training. This encouraged the model to learn more robust representations, less prone to overfitting.
- **Pooling Strategy**: Mean pooling still emerged as the best approach, indicating its consistency and suitability for capturing nuanced sentence-level semantics.
- **Outcome**: The additional dynamic masking objective led to better generalization, improved resilience to noise, and strong performance across validation metrics, while maintaining efficiency due to selective fine-tuning.

3. SBERT Embedding Alignment using Contrastive Loss (Bi-Encoder):



- **Base Model**: Similar SBERT architecture, with the top two encoder layers unfrozen for fine-tuning.
- **Selective Fine-Tuning**: Focusing updates on just the last two layers allowed the model to refine its representations specifically for the alignment task, building upon the stable, pre-trained foundation.
- **Contrastive Loss**: Replaced the conventional binary cross-entropy with a contrastive loss function to directly optimize the positioning of embeddings in semantic space. Similar question pairs were brought closer, while dissimilar pairs were pushed apart beyond a margin.
- **Performance Considerations**: While this method created a well-structured embedding space and was effective in principle, it required careful tuning of the margin parameter and did not outperform the strongest SBERT setups that combined mean pooling and auxiliary tasks.
- **Outcome**: Provided valuable insights into embedding space geometry, though ultimately was slightly less effective than the best-performing bi-encoder models without contrastive loss.

4. BERT Embedding Alignment using Contrastive Loss (Bi-Encoder):



- **Base Model**: Adopted a standard BERT-base-uncased model, which is larger (12 layers) and more computationally expensive than SBERT.
- **Selective Fine-Tuning**: As with the SBERT variants, only the last two layers of BERT were unfrozen, preserving general language understanding while fine-tuning for question similarity alignment.
- **Contrastive Loss Implementation**: Identical contrastive training setup as with SBERT, aiming to improve semantic separation between similar and dissimilar question pairs.
- **Resource Implications**: While performance remained in the same range as the SBERT-based contrastive approach, the computational overhead was significantly higher. Training times increased, and hardware resource demands were greater due to BERT's larger architecture.
- **Outcome**: Demonstrated that a heavier BERT model does not necessarily outperform the lighter SBERT variant. The efficiency and comparable accuracy of SBERT make it the preferable choice under similar conditions.

5. Cross-Encoder BERT:

- **Base Model**: Employed BERT-base-uncased, feeding both questions simultaneously into the same encoder, rather than encoding them separately.

- **Selective Fine-Tuning**: Consistent with previous models, only the top two layers were unfrozen. This strategic approach retained the well-trained lower layers while allowing the top layers to adapt to joint question representation.
- **Joint Input Processing**: Instead of producing independent embeddings for each question, this architecture passed question pairs as a single combined sequence ([CLS] q1 [SEP] q2 [SEP]). The BERT attention mechanism thus directly modeled interactions between the two questions, capturing richer pairwise semantics.
- **Performance Gains**: Achieved the highest F1 scores among all tested configurations, benefiting from the integrated encoding process. The model's capacity to assess both questions simultaneously led to more accurate similarity judgments.
- **Computational Considerations**: This method was more resource-intensive, as each evaluation required reprocessing both questions. However, techniques like mixed-precision training and careful selective fine-tuning helped mitigate computational burdens.
- **Outcome**: Delivered top-tier performance at a higher computational cost, showcasing the trade-off between efficiency (bi-encoder) and accuracy (cross-encoder).

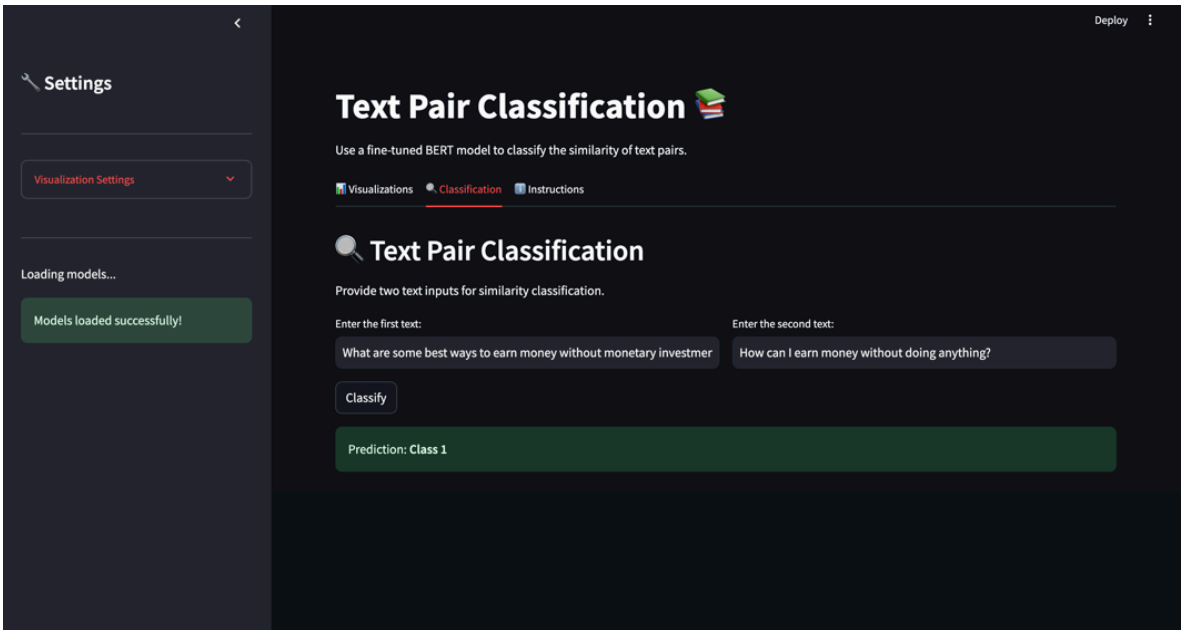
RESULTS

Algorithm	**F1-macro**
Naïve Bayes (tf-idf_weighted)	61.3
Logistic Regression (tf-idf_weighted)	74%
Random Forest (tf-idf_weighted)	81.7%
XGBoost (tf-idf_weighted)	**83%**
Naïve Bayes (BERT)	61.82%
Logistic Regression (BERT)	79.31%
Random Forest (BERT)	82.04%
XGBoost (BERT)	**84.10%**
GRU (BERT)	38.85%
LSTM (BERT)	39.93%
Siamese Bert + FCNN	**89.07%**
Siamese Bert + Dynamic Masking	**89.68%**
SBERT+ contrastive loss	80.48%
BERT + contrastive loss	79.48%
Cross-Encoder BERT	**93.28%**

Table 3: End-to-end performance table

Results Summary: In our final group project, we evaluated various algorithms to determine their effectiveness in identifying contextual similarities between questions. Among the traditional machine learning models utilizing TF-IDF weighted word embeddings, XGBoost achieved the highest F1-macro score of 83%, followed closely by Random Forest at 81.7%. For models using BERT embeddings, XGBoost again outperformed other models with an F1-macro score of 84.10%, indicating its robustness across different feature representations. Advanced deep learning models showed even more impressive results; the Cross-Encoder BERT model achieved the highest F1-macro score of 93.28%, significantly outperforming other approaches. Notably, the Siamese BERT models with FCNN and Dynamic Masking also performed exceptionally well, achieving F1-macro scores of 89.07% and 89.68% respectively. These results underscore the efficacy of leveraging transformer-based models and advanced neural network architectures for natural language processing tasks.

Streamlit App: Our final group project features a Streamlit app designed to determine the contextual similarity between two questions in seconds. The app also visualizes data using t-SNE plots in both 2D and 3D, providing an intuitive understanding of the question embeddings and their relationships.



Streamlit App Launch Window

References:

Libraries and Tools

- Python libraries for data processing, visualization, and NLP:
  - pandas: Pandas Documentation
  - numpy: NumPy Documentation
  - matplotlib: Matplotlib Documentation
  - seaborn: Seaborn Documentation
  - nltk: NLTK Documentation

Specific Algorithms and Techniques

- **TF-IDF Vectorization**:
  - tfidf_new.py module using TF-IDF weighted word embeddings.
  - TF-IDF Theory
- **Feature Engineering and Extraction**:
  - feature_extraction.py module calculating features such as Jaccard similarity, token-based ratios, and fuzzy matching.
  - Libraries:
    - fuzzywuzzy: FuzzyWuzzy GitHub
    - distance: Distance Library

Data Preprocessing

- pre_processing.py script for text normalization, stemming, and cleaning using:
  - NLTK's PorterStemmer and stopwords.
  - BeautifulSoup Documentation

Visualization

- visualise.py includes visualization methods such as violin plots and KL divergence:
  - Seaborn Violin Plot
  - Scipy KL Divergence

Model Splitting and Training

- train_test_split_and_random_model.py script uses sklearn for splitting data and evaluating baseline models:
  - Train-Test Split Documentation

General References for Techniques

- **NLP Techniques and Methods**:
    - Jurafsky, D., & Martin, J. H. (2022). *Speech and Language Processing* (3rd ed.). Pearson.

Dataset and Papers

- First Quora Dataset Release: Question Pairs
- Reimers, N. (2019). Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. arXiv preprint arXiv:1908.10084.
- Wang, F., & Liu, H. (2021). Understanding the behavior of contrastive loss. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition (pp. 2495-2504).
- Cross-Encoder Examples
- Yadav, G., Sinha, P., Sinha, P., & Jha, V. (2023). An Analytical Study on the Questions Comparing by Using Different Machine Learning Models with Special Reference to Random Forest, XGBoost, etc. MIET-Journal of Engineers and Professional Management & Allied Research, 96.
- PyTorch Data Tutorial

GitHub Repository Link: https://github.com/abhradeepd/NLP-Final-Project-Group-5