Model Architecture:

The implemented model leverages BERT (Bidirectional Encoder Representations from Transformers) for movie rating prediction. BERT is a pre-trained transformer-based model that excels in understanding contextual relationships within natural language. The model architecture consists of a simple linear layer (`nn.Linear`) that takes BERT embeddings as input and produces a single output, representing the predicted movie rating.

- **Tokenizer and Embedding:** The BERT tokenizer is utilized to preprocess movie titles, and the BERT model is employed to obtain embeddings. The get_embeddings function tokenizes the input text and retrieves the pooler output from the BERT model, which is then used as input to the linear regression layer.
- **Rating Predictor Neural Network:** The neural network, named RatingPredictor, has a single linear layer (nn.Linear) with an input size of 768 (size of BERT embeddings) and an output size of 1. The model is trained to predict movie ratings as a regression task.
- **Training:** The model is trained using mean squared error (MSE) loss and optimized with the Adam optimizer. The training loop iterates over the training data, tokenizes movie titles, computes BERT embeddings, makes predictions, calculates the loss, performs backpropagation, and updates the model parameters.

```
class RatingPredictor(nn.Module):
  def __init__(self, input_size):
    super(RatingPredictor, self).__init__()
    self.fc = nn.Linear(input_size, 1)

  def forward(self, x):
    x = x.view(x.size(0), -1)  # Flatten the input if needed
    return self.fc(x)
```

The forward function flattens the BERT embeddings and passes them through the linear layer, producing a continuous output that represents the predicted movie rating.

CHOICE OF DATASET:-

The dataset used for training and testing the model is derived from the MovieLens 100k dataset. It comprises information about movie ratings given by users, including user IDs, movie IDs, ratings, and timestamps. The dataset is split into training and testing sets, with 80% of the data used for training and 20% for testing.

## Challenges Faced During Implementation

1. **Data Preprocessing:** The dataset required careful preprocessing to merge movie information with rating data and create a suitable input format for the model.

2. **Tokenization and Embedding:** The BERT model necessitates tokenization and embedding of the movie titles, which introduces additional complexities and requires appropriate handling.

3. **Model Training:** Training a model with BERT embeddings involves adjusting hyperparameters such as learning rate, epochs, and dealing with potential overfitting challenges.

## Results of Model Evaluation

The model's performance is evaluated on the test set using the Root Mean Squared Error (RMSE) metric. The predictions are compared against the actual ratings, and the RMSE is calculated to measure the model's accuracy.

```
rating_predictor_model.eval()
predictions = []
labels = []
for _, row in test_data.iterrows():
    text = row['title']
    rating = row['rating']

    # Tokenization and embedding
    embedding = get_embeddings(text)

    # Prediction
    output = rating_predictor_model(embedding)
    predictions.append(output.item())
    labels.append(rating)

rmse = ((torch.tensor(predictions) - torch.tensor(labels)) ** 2).mean().sqrt()
print(f"RMSE: {rmse}")
```

The printed predictions and labels provide insight into the model's individual performance on the test set, and the RMSE gives an overall measure of the model's predictive accuracy.

This report outlines the key components of the implemented movie rating prediction model, the dataset used, challenges faced during implementation, and the results obtained through model evaluation.