Importing Libraries

In [1]:
```
from keras.datasets import imdb
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers import Embedding, LSTM, Dense, Dropout, Bidirectional, BatchNo
from keras.optimizers import AdamW
from keras.callbacks import EarlyStopping, ModelCheckpoint
from keras.initializers import Constant
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import classification_report, precision_recall_curve, confu
```

Loading IMDB Dataset

In [2]:
```
# Load IMDB dataset
VOCABULARY_SIZE = 10000
(MAX_SEQUENCE_LENGTH, EMBEDDING_DIM) = (150, 100)

(train_sequences, train_labels), (test_sequences, test_labels) = imdb.load_data(
```

In [3]:
```
# Pad sequences to a fixed length
train_sequences = pad_sequences(train_sequences, maxlen=MAX_SEQUENCE_LENGTH)
test_sequences = pad_sequences(test_sequences, maxlen=MAX_SEQUENCE_LENGTH)
```

In [4]:
```
# Define subsets for training (100, 500, 1000 samples)
SUBSET_SIZES = [100, 500, 1000]
subset_train_data = [train_sequences[:size] for size in SUBSET_SIZES]
subset_train_labels = [train_labels[:size] for size in SUBSET_SIZES]
```

Loading Pretrained GloVe Embeddings

In [5]:
```
# Load pretrained GloVe embeddings
glove_path = 'glove.6B.100d.txt'
embedding_index = {}
with open(glove_path, 'r') as glove_file:
    for line in glove_file:
        word, *coefficients = line.split()
        embedding_index[word] = np.array(coefficients, dtype='float32')
```

In [6]:
```
# Create embedding matrix
word_index = imdb.get_word_index()
embedding_matrix = np.zeros((VOCABULARY_SIZE, EMBEDDING_DIM))
for word, i in word_index.items():
    if i < VOCABULARY_SIZE:
        vector = embedding_index.get(word)
        if vector is not None:
            embedding_matrix[i] = vector
```

In [7]:
```python
# Attention layer for sequence-to-vector models
from keras.layers import Layer, Input
import tensorflow as tf

class Attention(Layer):
    def __init__(self, **kwargs):
        super(Attention, self).__init__(**kwargs)

    def build(self, input_shape):
        self.W = self.add_weight(shape=(input_shape[-1], input_shape[-1]),
                                 initializer='glorot_uniform', trainable=True)
        self.b = self.add_weight(shape=(input_shape[-1],),
                                 initializer='zeros', trainable=True)
        self.u = self.add_weight(shape=(input_shape[-1],),
                                 initializer='glorot_uniform', trainable=True)
        super(Attention, self).build(input_shape)

    def call(self, inputs):
        score = tf.nn.tanh(tf.tensordot(inputs, self.W, axes=[2, 0]) + self.b)
        score = tf.tensordot(score, self.u, axes=1)
        attention_weights = tf.nn.softmax(score, axis=1)
        context_vector = attention_weights[..., tf.newaxis] * inputs
        return tf.reduce_sum(context_vector, axis=1)
```

Custom RNN Models

In [8]:
```python
def build_deep_rnn(use_pretrained_embedding=False):
    model = Sequential()
    if use_pretrained_embedding:
        embedding_layer = Embedding(VOCABULARY_SIZE, EMBEDDING_DIM,
                                    embeddings_initializer=Constant(embedding_ma
                                    input_length=MAX_SEQUENCE_LENGTH,
                                    trainable=True)
    else:
        embedding_layer = Embedding(VOCABULARY_SIZE, EMBEDDING_DIM, input_length

    model.add(embedding_layer)
    model.add(Bidirectional(LSTM(128, return_sequences=True)))
    model.add(Dropout(0.5))
    model.add(BatchNormalization())
    model.add(Attention())
    model.add(Dense(64, activation='relu'))
    model.add(Dropout(0.4))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(optimizer=AdamW(learning_rate=1e-4), loss='binary_crossentropy
    return model
```

In [9]:
```python
from keras.callbacks import ModelCheckpoint, EarlyStopping

# Callback for early stopping and saving the best model
early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weig
model_checkpoint = ModelCheckpoint('best_model.keras', save_best_only=True, moni
```

Training and Evaluation

In [10]:
```
# Function to train and evaluate a model
def train_and_evaluate_model(data, labels, test_data, test_labels, model_type):
    model = build_deep_rnn(use_pretrained_embedding=(model_type == "pretrained")
    model.build(input_shape=(None, 150))
    history = model.fit(data, labels, epochs=20, batch_size=32, validation_data=
                        callbacks=[early_stopping, model_checkpoint])
    evaluation = model.evaluate(test_data, test_labels)
    return model, history, evaluation
```

In [11]:
```
pre_trained_model = build_deep_rnn(use_pretrained_embedding=True)
pre_trained_model.build(input_shape=(None, 150))  # Specify the input shape
print(pre_trained_model.summary())

custom_rnn = build_deep_rnn(use_pretrained_embedding=False)
custom_rnn.build(input_shape=(None, 150))  # Specify the input shape
print(custom_rnn.summary())
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/embedding.py:90: Us
erWarning: Argument `input_length` is deprecated. Just remove it.
  warnings.warn(
```

Model: "sequential"

Layer (type)	Output Shape	
embedding (Embedding)	(None, 150, 100)	
bidirectional (Bidirectional)	(None, 150, 256)	
dropout (Dropout)	(None, 150, 256)	
batch_normalization (BatchNormalization)	(None, 150, 256)	
attention (Attention)	(None, 256)	
dense (Dense)	(None, 64)	
dropout_1 (Dropout)	(None, 64)	
dense_1 (Dense)	(None, 1)	

Total params: 1,318,081 (5.03 MB)

Trainable params: 1,317,569 (5.03 MB)

Non-trainable params: 512 (2.00 KB)

None

Model: "sequential_1"

Layer (type)	Output Shape	
embedding_1 (Embedding)	(None, 150, 100)	
bidirectional_1 (Bidirectional)	(None, 150, 256)	
dropout_2 (Dropout)	(None, 150, 256)	
batch_normalization_1 (BatchNormalization)	(None, 150, 256)	
attention_1 (Attention)	(None, 256)	
dense_2 (Dense)	(None, 64)	
dropout_3 (Dropout)	(None, 64)	
dense_3 (Dense)	(None, 1)	

Total params: 1,318,081 (5.03 MB)

Trainable params: 1,317,569 (5.03 MB)

Non-trainable params: 512 (2.00 KB)

None

Comparison and Plotting

In [12]:
```
# **Plotting Function**

def plot_metrics(history, model_name, sample_size):
    """
    Plot training and validation accuracy/loss for a given model and dataset siz

    Parameters:
        history (History): Training history object from Keras.
        model_name (str): Name of the model being trained.
        sample_size (int): Number of training samples used.
    """
    fig, ax = plt.subplots(1, 2, figsize=(16, 6))

    # Accuracy Plot
    ax[0].plot(history.history['accuracy'], label='Training Accuracy', marker='o
    ax[0].plot(history.history['val_accuracy'], label='Validation Accuracy', mar
    ax[0].set_title(f'{model_name} Accuracy (Sample Size: {sample_size})', fonts
    ax[0].set_xlabel('Epochs', fontsize=12)
    ax[0].set_ylabel('Accuracy', fontsize=12)
    ax[0].legend(fontsize=10)
    ax[0].grid(True, linestyle='--', alpha=0.6)

    # Loss Plot
    ax[1].plot(history.history['loss'], label='Training Loss', marker='o', color
    ax[1].plot(history.history['val_loss'], label='Validation Loss', marker='s',
    ax[1].set_title(f'{model_name} Loss (Sample Size: {sample_size})', fontsize=
    ax[1].set_xlabel('Epochs', fontsize=12)
    ax[1].set_ylabel('Loss', fontsize=12)
```

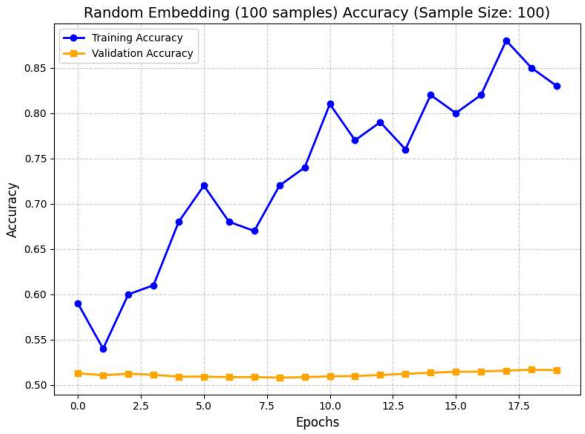
```
    ax[1].legend(fontsize=10)
    ax[1].grid(True, linestyle='--', alpha=0.6)

    plt.tight_layout()
    plt.show()
```

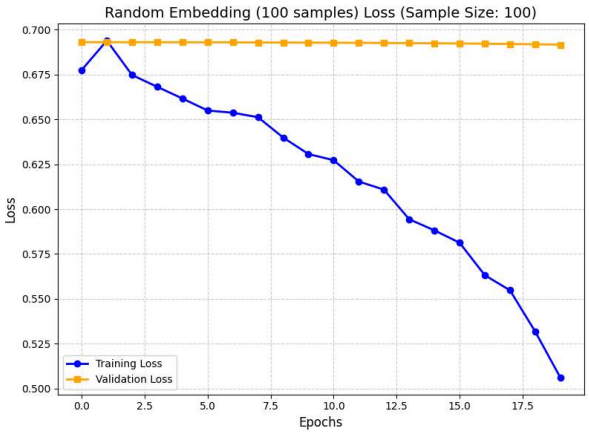
In [13]:

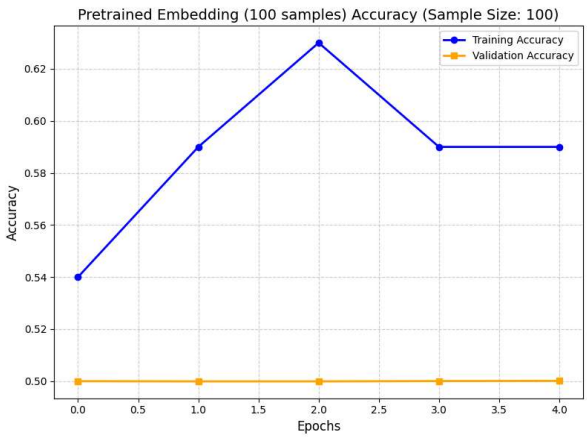
```
# Store results
results = []

for size, data, labels in zip(SUBSET_SIZES, subset_train_data, subset_train_labe
    print(f"Training on subset size: {size}")

    # Random Embedding Model
    model_random, history_random, eval_random = train_and_evaluate_model(data, l

    # Pretrained Embedding Model
    model_pretrained, history_pretrained, eval_pretrained = train_and_evaluate_m

    # Store results
    results.append({
        'Sample Size': size,
        'Random Embedding Accuracy': eval_random[1],
        'Pretrained Embedding Accuracy': eval_pretrained[1]
    })

    # Plot metrics
    plot_metrics(history_random, f'Random Embedding ({size} samples)', size)
    plot_metrics(history_pretrained, f'Pretrained Embedding ({size} samples)', s
```

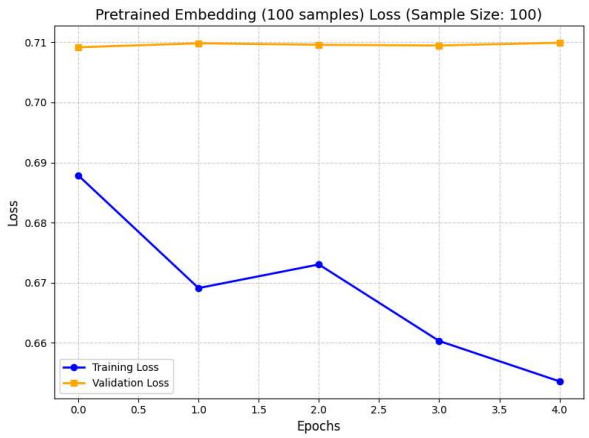
```
Training on subset size: 100
Epoch 1/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 16s 2s/step - accuracy: 0.6089 - loss: 0.6779 - val_accu
racy: 0.5130 - val_loss: 0.6931
Epoch 2/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 7s 2s/step - accuracy: 0.5358 - loss: 0.6985 - val_accur
acy: 0.5107 - val_loss: 0.6931
Epoch 3/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 6s 2s/step - accuracy: 0.5910 - loss: 0.6723 - val_accur
acy: 0.5125 - val_loss: 0.6930
Epoch 4/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 6s 2s/step - accuracy: 0.6013 - loss: 0.6694 - val_accur
acy: 0.5112 - val_loss: 0.6930
Epoch 5/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 10s 3s/step - accuracy: 0.7043 - loss: 0.6584 - val_accu
racy: 0.5092 - val_loss: 0.6930
Epoch 6/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 16s 2s/step - accuracy: 0.7182 - loss: 0.6540 - val_accu
racy: 0.5092 - val_loss: 0.6930
Epoch 7/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 9s 2s/step - accuracy: 0.7168 - loss: 0.6509 - val_accur
acy: 0.5086 - val_loss: 0.6930
Epoch 8/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 10s 2s/step - accuracy: 0.7024 - loss: 0.6511 - val_accu
racy: 0.5087 - val_loss: 0.6929
Epoch 9/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 11s 2s/step - accuracy: 0.6745 - loss: 0.6471 - val_accu
racy: 0.5081 - val_loss: 0.6929
Epoch 10/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 11s 2s/step - accuracy: 0.7377 - loss: 0.6257 - val_accu
racy: 0.5086 - val_loss: 0.6929
Epoch 11/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 9s 2s/step - accuracy: 0.7771 - loss: 0.6301 - val_accur
acy: 0.5095 - val_loss: 0.6928
Epoch 12/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 7s 2s/step - accuracy: 0.8059 - loss: 0.6154 - val_accur
acy: 0.5098 - val_loss: 0.6927
Epoch 13/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 5s 2s/step - accuracy: 0.7879 - loss: 0.6157 - val_accur
acy: 0.5110 - val_loss: 0.6927
Epoch 14/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 10s 2s/step - accuracy: 0.7748 - loss: 0.5832 - val_accu
racy: 0.5124 - val_loss: 0.6926
Epoch 15/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 11s 2s/step - accuracy: 0.8030 - loss: 0.5895 - val_accu
racy: 0.5135 - val_loss: 0.6925
Epoch 16/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 5s 2s/step - accuracy: 0.7950 - loss: 0.5859 - val_accur
acy: 0.5145 - val_loss: 0.6924
Epoch 17/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 10s 2s/step - accuracy: 0.7999 - loss: 0.5627 - val_accu
racy: 0.5148 - val_loss: 0.6922
Epoch 18/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 10s 3s/step - accuracy: 0.9176 - loss: 0.5480 - val_accu
racy: 0.5159 - val_loss: 0.6921
Epoch 19/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 16s 2s/step - accuracy: 0.8702 - loss: 0.5251 - val_accu
racy: 0.5168 - val_loss: 0.6919
Epoch 20/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 11s 2s/step - accuracy: 0.8310 - loss: 0.5025 - val_accu
```

racy: 0.5164 - val_loss: 0.6917
782/782 ━━━━━━━━━━━━━━━━━━━━ **6s** 8ms/step - accuracy: 0.5227 - loss: 0.6914
Epoch 1/20
4/4 ━━━━━━━━━━━━━━━━━━━━ **10s** 2s/step - accuracy: 0.5441 - loss: 0.6916 - val_accu
racy: 0.5000 - val_loss: 0.7092
Epoch 2/20
4/4 ━━━━━━━━━━━━━━━━━━━━ **11s** 2s/step - accuracy: 0.6089 - loss: 0.6647 - val_accu
racy: 0.4999 - val_loss: 0.7098
Epoch 3/20
4/4 ━━━━━━━━━━━━━━━━━━━━ **6s** 2s/step - accuracy: 0.6260 - loss: 0.6735 - val_accur
acy: 0.4999 - val_loss: 0.7096
Epoch 4/20
4/4 ━━━━━━━━━━━━━━━━━━━━ **10s** 2s/step - accuracy: 0.6120 - loss: 0.6492 - val_accu
racy: 0.5000 - val_loss: 0.7095
Epoch 5/20
4/4 ━━━━━━━━━━━━━━━━━━━━ **6s** 2s/step - accuracy: 0.5683 - loss: 0.6556 - val_accur
acy: 0.5001 - val_loss: 0.7099
782/782 ━━━━━━━━━━━━━━━━━━━━ **7s** 9ms/step - accuracy: 0.5073 - loss: 0.7064



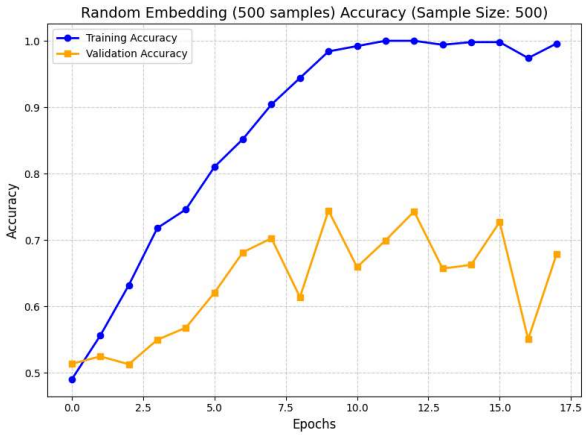


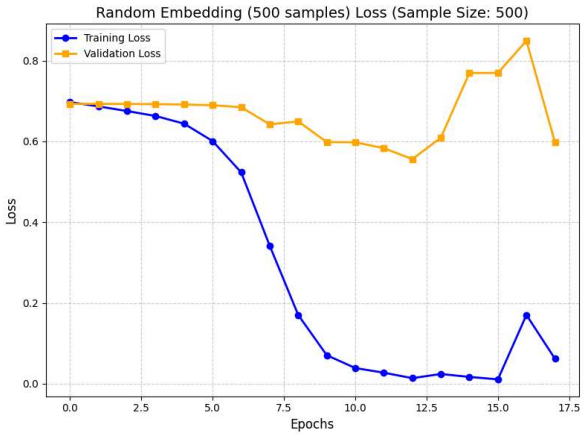


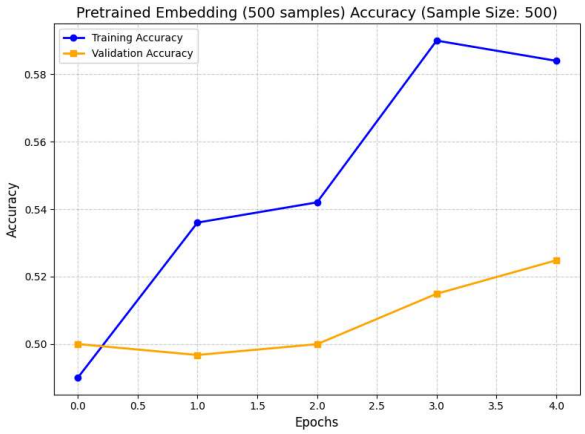


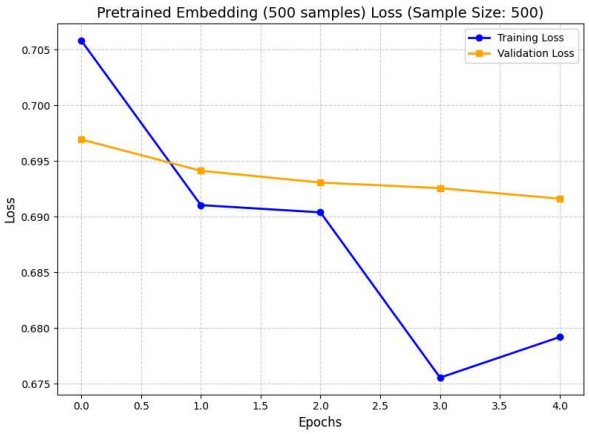
```
Training on subset size: 500
Epoch 1/20
16/16 ━━━━━━━━━━━━━━━━━━━━ 10s 496ms/step - accuracy: 0.4814 - loss: 0.6981 - val
_accuracy: 0.5134 - val_loss: 0.6931
Epoch 2/20
16/16 ━━━━━━━━━━━━━━━━━━━━ 9s 431ms/step - accuracy: 0.5465 - loss: 0.6889 - val_
accuracy: 0.5246 - val_loss: 0.6930
Epoch 3/20
16/16 ━━━━━━━━━━━━━━━━━━━━ 10s 389ms/step - accuracy: 0.6087 - loss: 0.6795 - val
_accuracy: 0.5128 - val_loss: 0.6927
Epoch 4/20
16/16 ━━━━━━━━━━━━━━━━━━━━ 10s 395ms/step - accuracy: 0.6928 - loss: 0.6680 - val
_accuracy: 0.5497 - val_loss: 0.6923
Epoch 5/20
16/16 ━━━━━━━━━━━━━━━━━━━━ 11s 464ms/step - accuracy: 0.7486 - loss: 0.6459 - val
_accuracy: 0.5679 - val_loss: 0.6914
Epoch 6/20
16/16 ━━━━━━━━━━━━━━━━━━━━ 10s 464ms/step - accuracy: 0.8408 - loss: 0.6023 - val
_accuracy: 0.6204 - val_loss: 0.6896
Epoch 7/20
16/16 ━━━━━━━━━━━━━━━━━━━━ 9s 402ms/step - accuracy: 0.8295 - loss: 0.5437 - val_
accuracy: 0.6815 - val_loss: 0.6847
Epoch 8/20
16/16 ━━━━━━━━━━━━━━━━━━━━ 10s 400ms/step - accuracy: 0.9057 - loss: 0.3929 - val
_accuracy: 0.7024 - val_loss: 0.6423
Epoch 9/20
16/16 ━━━━━━━━━━━━━━━━━━━━ 11s 461ms/step - accuracy: 0.9447 - loss: 0.1856 - val
_accuracy: 0.6138 - val_loss: 0.6496
Epoch 10/20
16/16 ━━━━━━━━━━━━━━━━━━━━ 6s 386ms/step - accuracy: 0.9890 - loss: 0.0683 - val_
accuracy: 0.7443 - val_loss: 0.5983
Epoch 11/20
16/16 ━━━━━━━━━━━━━━━━━━━━ 10s 399ms/step - accuracy: 0.9919 - loss: 0.0441 - val
_accuracy: 0.6592 - val_loss: 0.5979
Epoch 12/20
16/16 ━━━━━━━━━━━━━━━━━━━━ 11s 710ms/step - accuracy: 1.0000 - loss: 0.0261 - val
_accuracy: 0.6993 - val_loss: 0.5834
Epoch 13/20
16/16 ━━━━━━━━━━━━━━━━━━━━ 16s 394ms/step - accuracy: 1.0000 - loss: 0.0133 - val
_accuracy: 0.7429 - val_loss: 0.5561
Epoch 14/20
16/16 ━━━━━━━━━━━━━━━━━━━━ 11s 703ms/step - accuracy: 0.9957 - loss: 0.0185 - val
_accuracy: 0.6569 - val_loss: 0.6093
Epoch 15/20
16/16 ━━━━━━━━━━━━━━━━━━━━ 16s 381ms/step - accuracy: 0.9975 - loss: 0.0226 - val
_accuracy: 0.6625 - val_loss: 0.7694
Epoch 16/20
16/16 ━━━━━━━━━━━━━━━━━━━━ 11s 433ms/step - accuracy: 0.9970 - loss: 0.0128 - val
_accuracy: 0.7271 - val_loss: 0.7696
Epoch 17/20
16/16 ━━━━━━━━━━━━━━━━━━━━ 6s 370ms/step - accuracy: 0.9818 - loss: 0.1733 - val_
accuracy: 0.5503 - val_loss: 0.8492
Epoch 18/20
16/16 ━━━━━━━━━━━━━━━━━━━━ 10s 379ms/step - accuracy: 0.9991 - loss: 0.0523 - val
_accuracy: 0.6791 - val_loss: 0.5978
782/782 ━━━━━━━━━━━━━━━━━━━━ 7s 9ms/step - accuracy: 0.7368 - loss: 0.5596
Epoch 1/20
16/16 ━━━━━━━━━━━━━━━━━━━━ 9s 405ms/step - accuracy: 0.4848 - loss: 0.7051 - val_
accuracy: 0.5000 - val_loss: 0.6969
Epoch 2/20
```

```
16/16 ━━━━━━━━━━━━━━━━━━━━ 10s 380ms/step - accuracy: 0.5371 - loss: 0.6840 - val
_accuracy: 0.4967 - val_loss: 0.6941
Epoch 3/20
16/16 ━━━━━━━━━━━━━━━━━━━━ 11s 426ms/step - accuracy: 0.5502 - loss: 0.6934 - val
_accuracy: 0.4999 - val_loss: 0.6931
Epoch 4/20
16/16 ━━━━━━━━━━━━━━━━━━━━ 6s 372ms/step - accuracy: 0.6076 - loss: 0.6690 - val_
accuracy: 0.5149 - val_loss: 0.6926
Epoch 5/20
16/16 ━━━━━━━━━━━━━━━━━━━━ 10s 380ms/step - accuracy: 0.6219 - loss: 0.6687 - val
_accuracy: 0.5248 - val_loss: 0.6916
782/782 ━━━━━━━━━━━━━━━━━━━━ 8s 10ms/step - accuracy: 0.4927 - loss: 0.6981
```

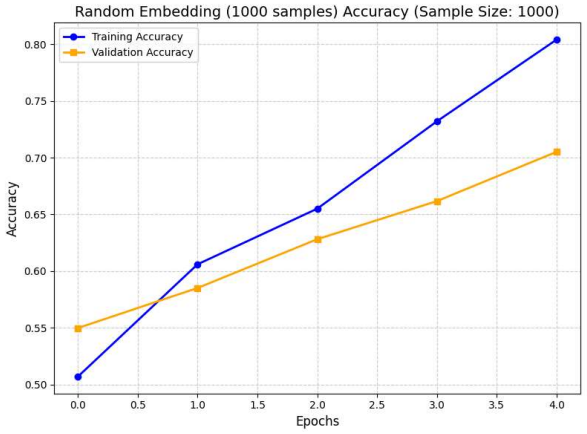


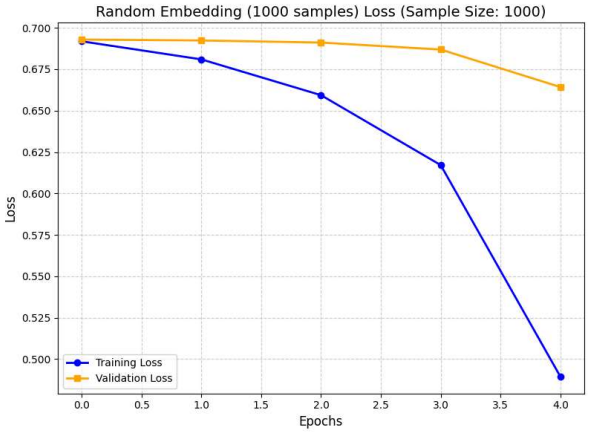


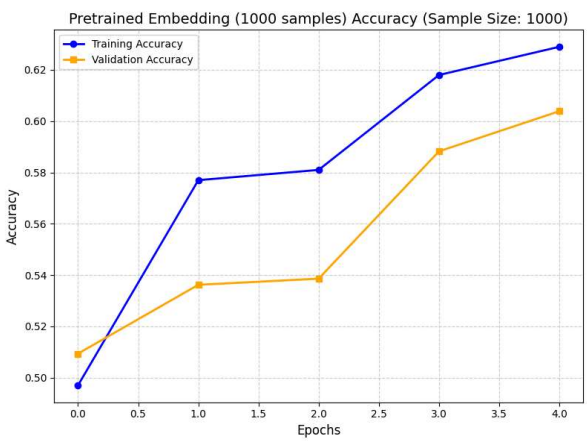


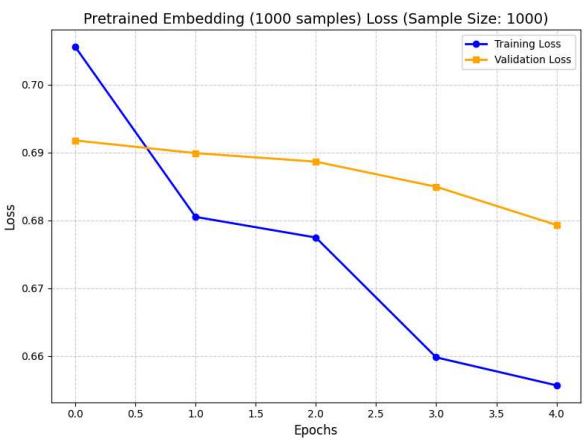


```
Training on subset size: 1000
Epoch 1/20
32/32 ━━━━━━━━━━━━━━━━━━━━ 14s 366ms/step - accuracy: 0.4750 - loss: 0.6960 - val
_accuracy: 0.5498 - val_loss: 0.6929
Epoch 2/20
32/32 ━━━━━━━━━━━━━━━━━━━━ 6s 195ms/step - accuracy: 0.6002 - loss: 0.6835 - val_
accuracy: 0.5851 - val_loss: 0.6924
Epoch 3/20
32/32 ━━━━━━━━━━━━━━━━━━━━ 10s 193ms/step - accuracy: 0.6736 - loss: 0.6592 - val
_accuracy: 0.6281 - val_loss: 0.6911
Epoch 4/20
32/32 ━━━━━━━━━━━━━━━━━━━━ 7s 227ms/step - accuracy: 0.7378 - loss: 0.6254 - val_
accuracy: 0.6617 - val_loss: 0.6869
Epoch 5/20
32/32 ━━━━━━━━━━━━━━━━━━━━ 9s 194ms/step - accuracy: 0.7520 - loss: 0.5416 - val_
accuracy: 0.7051 - val_loss: 0.6642
782/782 ━━━━━━━━━━━━━━━━━━━━ 8s 10ms/step - accuracy: 0.5507 - loss: 0.6929
Epoch 1/20
32/32 ━━━━━━━━━━━━━━━━━━━━ 11s 244ms/step - accuracy: 0.4854 - loss: 0.7081 - val
_accuracy: 0.5094 - val_loss: 0.6918
Epoch 2/20
32/32 ━━━━━━━━━━━━━━━━━━━━ 14s 353ms/step - accuracy: 0.5771 - loss: 0.6757 - val
_accuracy: 0.5362 - val_loss: 0.6899
Epoch 3/20
32/32 ━━━━━━━━━━━━━━━━━━━━ 16s 198ms/step - accuracy: 0.5877 - loss: 0.6803 - val
_accuracy: 0.5386 - val_loss: 0.6886
Epoch 4/20
32/32 ━━━━━━━━━━━━━━━━━━━━ 11s 231ms/step - accuracy: 0.6395 - loss: 0.6551 - val
_accuracy: 0.5882 - val_loss: 0.6850
Epoch 5/20
32/32 ━━━━━━━━━━━━━━━━━━━━ 14s 353ms/step - accuracy: 0.6344 - loss: 0.6584 - val
_accuracy: 0.6039 - val_loss: 0.6793
782/782 ━━━━━━━━━━━━━━━━━━━━ 7s 8ms/step - accuracy: 0.5154 - loss: 0.6914
```



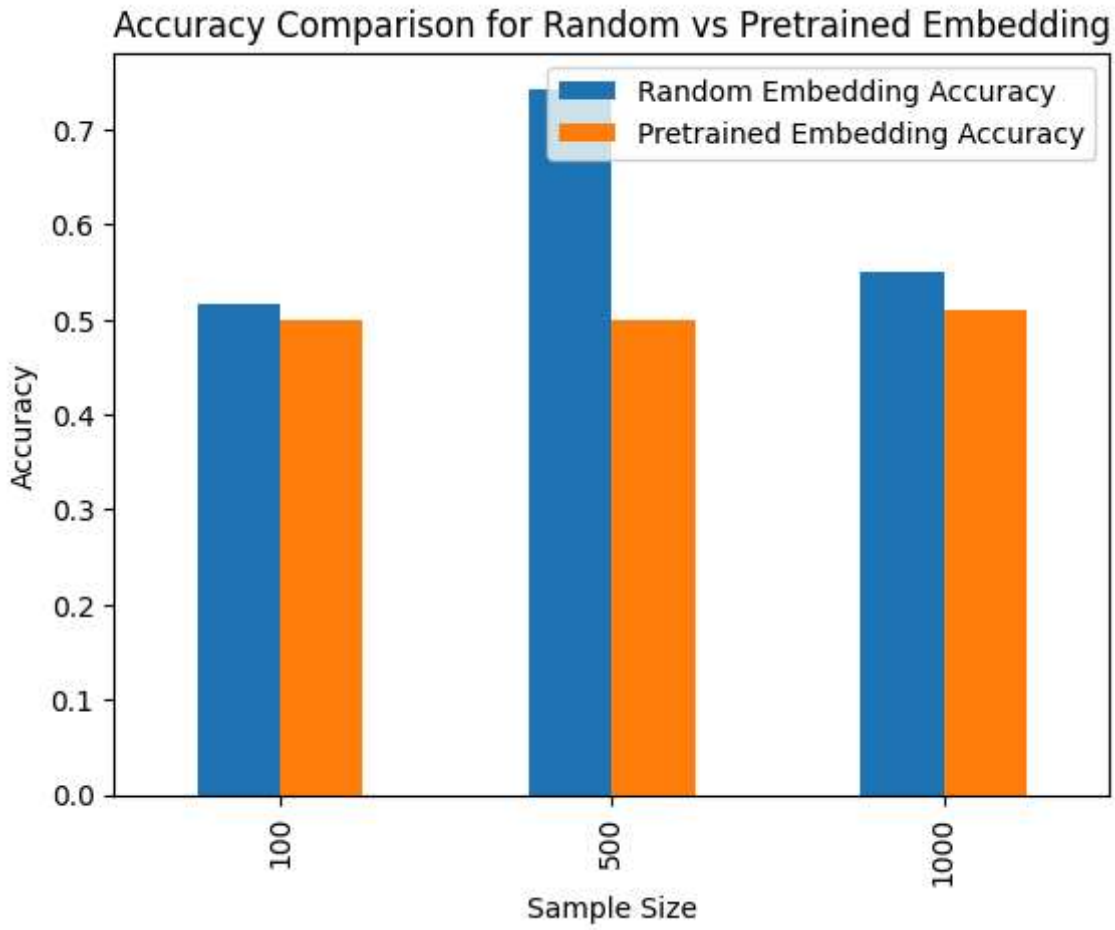






Model Comparison

In [14]:
```
# Convert results to DataFrame
df_results = pd.DataFrame(results)

# Plot final comparison
df_results.plot(x='Sample Size', y=['Random Embedding Accuracy', 'Pretrained Emb
plt.title("Accuracy Comparison for Random vs Pretrained Embedding")
plt.ylabel("Accuracy")
plt.show()
```



In [15]:
```
df_results
```

Out[15]:

	Sample Size	Random Embedding Accuracy	Pretrained Embedding Accuracy
0	100	0.51640	0.50000
1	500	0.74292	0.49996
2	1000	0.54984	0.50936