In [ ]:
```
import pandas as pd
import nltk
from collections import Counter
import pandas as pd
from nltk.corpus import stopwords
import re
import seaborn as sns
import matplotlib.pyplot as plt
from wordcloud import WordCloud
import string
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense, Dropout, GlobalA
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv1D
from tensorflow.keras.layers import GlobalMaxPooling1D
```

In [3]:
```
# Load the training and test data
train_df = pd.read_csv("../data/nlp/train.csv")
test_df = pd.read_csv("../data/nlp/test.csv")
```

In [4]:
```
# Show info
print("Train shape:", train_df.shape)
print("\nTest shape:", test_df.shape)

# Preview the data
print('\nPreview the first few rows:')
display(train_df.head())
```

```
Train shape: (7613, 5)

Test shape: (3263, 4)

Preview the first few rows:
```

	id	keyword	location	text	target
0	1	NaN	NaN	Our Deeds are the Reason of this #earthquake M...	1
1	4	NaN	NaN	Forest fire near La Ronge Sask. Canada	1
2	5	NaN	NaN	All residents asked to 'shelter in place' are ...	1
3	6	NaN	NaN	13,000 people receive #wildfires evacuation or...	1
4	7	NaN	NaN	Just got sent this photo from Ruby #Alaska as ...	1

In [5]:
```
# Check for missing values
missing_values = train_df.isnull().sum()
print("\nMissing values:\n", missing_values)
```

```
Missing values:
 id             0
keyword       61
location    2533
text           0
target         0
dtype: int64
```

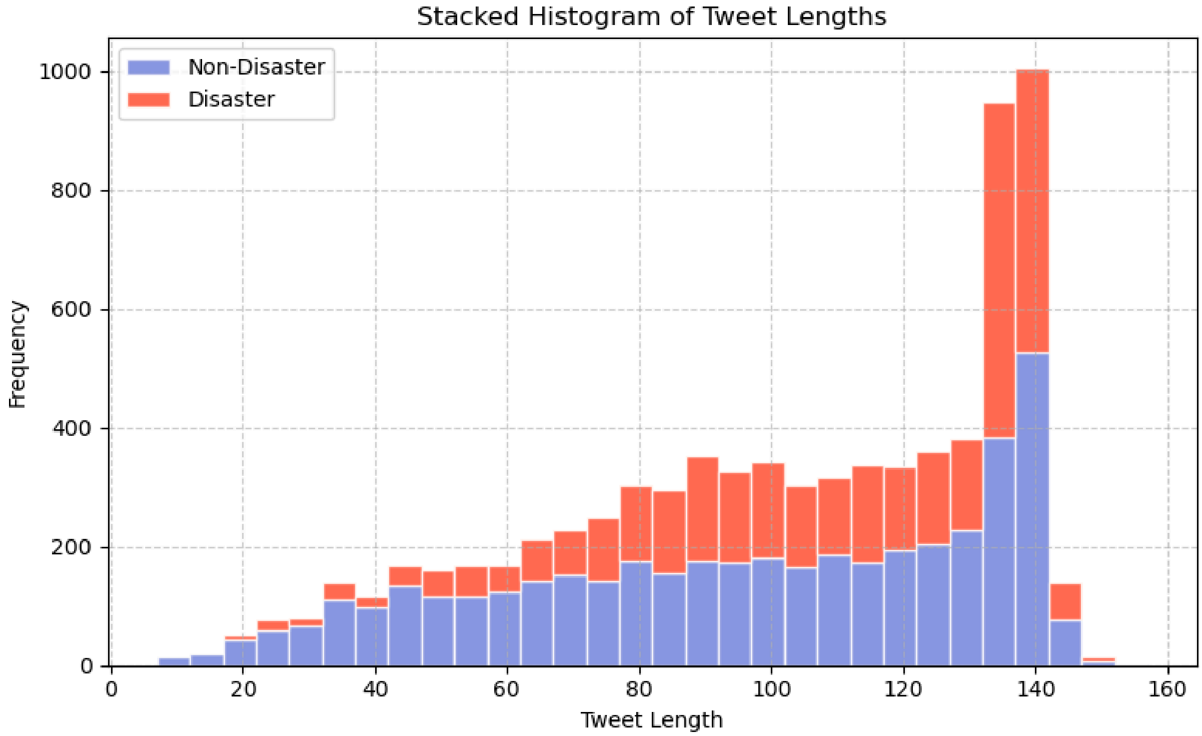
In [6]:
```
# Class balance
class_counts = train_df['target'].value_counts()
print("\nClass distribution:\n", class_counts)

# Percentage distribution
print("\nClass percentage:\n", class_counts / len(train_df) * 100)
```

```
Class distribution:
 target
0    4342
1    3271
Name: count, dtype: int64

Class percentage:
 target
0    57.034021
1    42.965979
Name: count, dtype: float64
```

In [7]:
```
print("Unique keywords:", train_df['keyword'].nunique())
print("Unique locations:", train_df['location'].nunique())
```

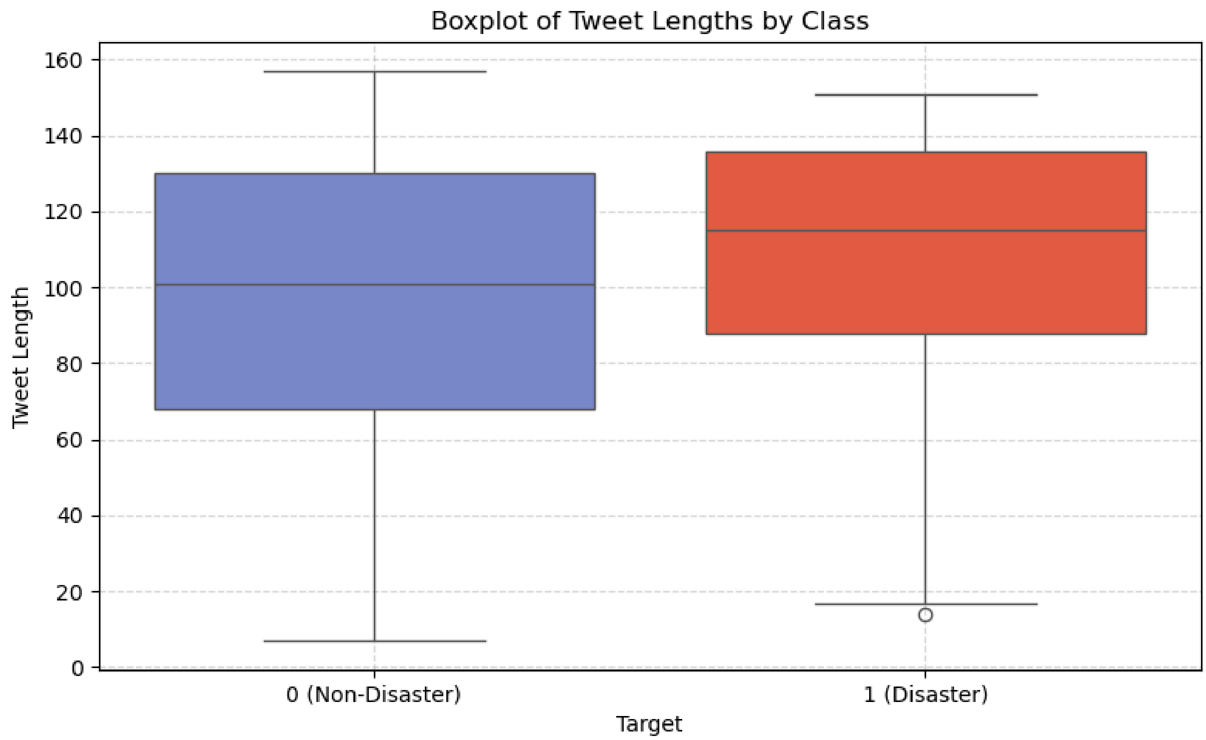
```
Unique keywords: 221
Unique locations: 3341
```

In [8]:
```
# Tweet lengths
train_df['text_len'] = train_df['text'].apply(len)

# Plot stacked histogram
plt.figure(figsize=(8, 5))
plt.hist(
    [train_df[train_df['target'] == 0]['text_len'], train_df[train_df['targe
    bins=30,
    stacked=True,
    label=['Non-Disaster', 'Disaster'],
    color=['#6C7EDA', '#FF4929'],
    edgecolor='white',
    alpha=0.8
)

plt.xlabel('Tweet Length')
plt.ylabel('Frequency')
plt.title('Stacked Histogram of Tweet Lengths')
plt.legend()
plt.grid(True, linestyle='--', alpha=0.6)
plt.tight_layout()
plt.show()
```



In [9]:
```
# Summary statistics by class
summary_stats = train_df.groupby('target')['text_len'].describe()
print("Summary Statistics:\n")
print(summary_stats)

# Boxplot
plt.figure(figsize=(8, 5))
sns.boxplot(
    x='target',
    y='text_len',
    data=train_df,
    hue='target',
    palette={0: '#6C7EDA', 1: '#FF4929'},
    dodge=False,
    legend=False
)

plt.xlabel('Target')
plt.ylabel('Tweet Length')
plt.title('Boxplot of Tweet Lengths by Class')
plt.xticks(ticks=[0, 1], labels=['0 (Non-Disaster)', '1 (Disaster)'])
plt.grid(True, linestyle='--', alpha=0.5)
plt.tight_layout()
plt.show()
```

```
Summary Statistics:

         count        mean        std   min   25%    50%    75%    max
target
0       4342.0   95.706817  35.885924   7.0  68.0  101.0  130.0  157.0
1       3271.0  108.113421  29.309854  14.0  88.0  115.0  136.0  151.0
```



In [10]:
```
# Download stopwords
stop_words = set(stopwords.words('english'))

def basic_tokenize(text):
    # Lowercase and remove punctuation using regex
    text = re.sub(r"[^\w\s]", "", text.lower())
    tokens = text.split()
    return [word for word in tokens if word not in stop_words and len(word)


# Tokenize by class
disaster_tokens = train_df[train_df['target'] == 1]['text'].apply(basic_toke
nondisaster_tokens = train_df[train_df['target'] == 0]['text'].apply(basic_t

# Get top 30 words
disaster_counts = Counter(disaster_tokens)
nondisaster_counts = Counter(nondisaster_tokens)

top30_disaster = pd.DataFrame(disaster_counts.most_common(30), columns=['wor
top30_nondisaster = pd.DataFrame(nondisaster_counts.most_common(30), columns

# Display
print("Top 30 Words in Disaster Tweets:")
display(top30_disaster)

print("Top 30 Words in Non-Disaster Tweets:")
display(top30_nondisaster)
```

Top 30 Words in Disaster Tweets:

	word	count
0	fire	178
1	news	136
2	via	121
3	disaster	117
4	california	111
5	suicide	110
6	police	107
7	amp	106
8	people	105
9	killed	93
10	like	92
11	hiroshima	86
12	storm	85
13	crash	84
14	fires	84
15	families	81
16	train	79
17	emergency	76
18	buildings	75
19	bomb	74
20	two	71
21	mh370	71
22	nuclear	70
23	attack	69
24	video	69
25	wildfire	69
26	get	66
27	accident	66
28	bombing	66
29	one	65

```
Top 30 Words in Non-Disaster Tweets:
```

	word	count
0	like	253
1	amp	192
2	new	168
3	get	163
4	dont	141
5	one	127
6	body	112
7	via	99
8	would	97
9	video	96
10	people	91
11	love	89
12	know	85
13	back	84
14	time	83
15	got	83
16	see	82
17	cant	81
18	emergency	81
19	full	81
20	day	78
21	youtube	76
22	going	75
23	still	72
24	fire	72
25	want	67
26	good	67
27	think	66
28	man	62
29	world	62

In [11]:
```
# Generate word clouds
wordcloud_disaster = WordCloud(width=800, height=400, background_color='whit
wordcloud_nondisaster = WordCloud(width=800, height=400, background_color='w

# Plot side by side
plt.figure(figsize=(16, 7))

plt.subplot(1, 2, 1)
plt.imshow(wordcloud_disaster, interpolation='bilinear')
plt.axis('off')
plt.title('Disaster Tweets Word Cloud', fontsize=16)

plt.subplot(1, 2, 2)
plt.imshow(wordcloud_nondisaster, interpolation='bilinear')
plt.axis('off')
plt.title('Non-Disaster Tweets Word Cloud', fontsize=16)

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

In [12]:
```
keyword_target = train_df.groupby('keyword')['target'].mean().sort_values(as
print(keyword_target.head(10))
print(keyword_target.tail(10))
```

```
keyword
debris                  1.000000
wreckage                1.000000
derailment              1.000000
outbreak                0.975000
oil%20spill             0.973684
typhoon                 0.973684
suicide%20bombing       0.969697
suicide%20bomber        0.967742
bombing                 0.931034
rescuers                0.914286
Name: target, dtype: float64
keyword
panicking      0.060606
blew%20up      0.060606
traumatised    0.057143
screaming      0.055556
electrocute    0.031250
body%20bag     0.030303
blazing        0.029412
ruin           0.027027
body%20bags    0.024390
aftershock     0.000000
Name: target, dtype: float64
```

In [ ]:

```python
# Drop missing keywords
keyword_target = train_df.dropna(subset=['keyword']).groupby('keyword')['tar

# Preview top keywords by frequency
print("Top 20 most frequent keywords:\n")
display(keyword_target.head(20))

# Top keywords most strongly associated with disasters
print("\nTop 10 keywords most associated with disasters:")
display(keyword_target.sort_values('mean', ascending=False).head(10))

# Top keywords most strongly associated with non-disasters
print("\nTop 10 keywords least associated with disasters:")
display(keyword_target.sort_values('mean', ascending=True).head(10))
```

```
Top 20 most frequent keywords:
```

	count	mean
keyword		
fatalities	45	0.577778
deluge	42	0.142857
armageddon	42	0.119048
sinking	41	0.195122
damage	41	0.463415
harm	41	0.097561
body%20bags	41	0.024390
outbreak	40	0.975000
evacuate	40	0.625000
fear	40	0.125000
collided	40	0.575000
siren	40	0.125000
twister	40	0.125000
windstorm	40	0.400000
sinkhole	39	0.692308
sunk	39	0.230769
hellfire	39	0.179487
weapon	39	0.358974
weapons	39	0.435897
famine	39	0.666667

```
Top 10 keywords most associated with disasters:
```

	count	mean
keyword		
wreckage	39	1.000000
derailment	39	1.000000
debris	37	1.000000
outbreak	40	0.975000
typhoon	38	0.973684
oil%20spill	38	0.973684
suicide%20bombing	33	0.969697
suicide%20bomber	31	0.967742
bombing	29	0.931034
rescuers	35	0.914286

```
Top 10 keywords least associated with disasters:
```

	count	mean
keyword		
aftershock	34	0.000000
body%20bags	41	0.024390
ruin	37	0.027027
blazing	34	0.029412
body%20bag	33	0.030303
electrocute	32	0.031250
screaming	36	0.055556
traumatised	35	0.057143
panicking	33	0.060606
blew%20up	33	0.060606

In [ ]:
```
top_locations = train_df['location'].value_counts().head(10)
print("Top locations:\n", top_locations)
```

```
Top locations:
 location
USA                104
New York            71
United States       50
London              45
Canada              29
Nigeria             28
UK                  27
Los Angeles, CA     26
India               24
Mumbai              22
Name: count, dtype: int64
```

In [ ]:
```
# Number and percentage of missing values
missing_locs = train_df['location'].isna().sum()
total_rows = len(train_df)
print(f"Missing locations: {missing_locs} / {total_rows} ({missing_locs / tc

# Number of unique locations
print("Unique locations:", train_df['location'].nunique())

# Most common locations
print("\nTop 10 most frequent locations:")
display(train_df['location'].value_counts().head(10))

# Location-target association
loc_target = (
    train_df.dropna(subset=['location'])
            .groupby('location')['target']
            .agg(['count', 'mean'])
            .sort_values(by='count', ascending=False)
)
print("\nTop 10 locations most strongly associated with disasters:")
display(loc_target[loc_target['count'] >= 10].sort_values('mean', ascending=

print("\nTop 10 locations least associated with disasters:")
display(loc_target[loc_target['count'] >= 10].sort_values('mean', ascending=
```

```
Missing locations: 2533 / 7613 (33.27%)
Unique locations: 3341

Top 10 most frequent locations:
location
USA                104
New York            71
United States       50
London              45
Canada              29
Nigeria             28
UK                  27
Los Angeles, CA     26
India               24
Mumbai              22
Name: count, dtype: int64
Top 10 locations most strongly associated with disasters:
```

location	count	mean
Mumbai	22	0.863636
India	24	0.833333
Nigeria	28	0.785714
Earth	11	0.727273
Washington, DC	21	0.714286
Sacramento, CA	10	0.700000
Washington, D.C.	13	0.692308
USA	104	0.644231
San Francisco, CA	11	0.636364
Worldwide	19	0.631579

```
Top 10 locations least associated with disasters:
```

location	count	mean
ss	10	0.100000
London, England	10	0.100000
NYC	12	0.166667
Everywhere	15	0.200000
Florida	14	0.214286
New York	71	0.225352
Kenya	20	0.250000
United Kingdom	14	0.285714
Texas	10	0.300000
Los Angeles, CA	26	0.307692

In [ ]:
```
# Download stopwords
stop_words = set(stopwords.words('english'))

def clean_text(text):
    text = text.lower()
    text = re.sub(r"http\S+", "", text)                    # remove URLs
    text = re.sub(r"@\w+|#\w+", "", text)                  # remove mentions
    text = re.sub(r"[^a-zA-Z0-9\s]", "", text)             # remove punctuati
    text = re.sub(r"\s+", " ", text).strip()               # remove extra whi
    tokens = text.split()
```

```
    tokens = [t for t in tokens if t not in stop_words]   # remove stopwords
    return " ".join(tokens)
```

In [17]:
```
train_df['clean_text'] = train_df['text'].apply(clean_text)
test_df['clean_text'] = test_df['text'].apply(clean_text)
```

In [18]:
```
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences

# Initialize tokenizer
tokenizer = Tokenizer()
tokenizer.fit_on_texts(train_df['clean_text'])

# Convert text to sequences
X_train_seq = tokenizer.texts_to_sequences(train_df['clean_text'])
X_test_seq = tokenizer.texts_to_sequences(test_df['clean_text'])

# Pad sequences
max_len = 100  # or use np.percentile([len(x) for x in X_train_seq], 95)
X_train_pad = pad_sequences(X_train_seq, maxlen=max_len, padding='post')
X_test_pad = pad_sequences(X_test_seq, maxlen=max_len, padding='post')

# Get vocab size
vocab_size = len(tokenizer.word_index) + 1
```

In [19]:
```
embedding_dim = 100
embedding_index = {}

# Path to glove.6B.100d.txt
glove_path = '../data/glove/glove.6B.100d.txt'

with open(glove_path, encoding='utf8') as f:
    for line in f:
        values = line.split()
        word = values[0]
        coefs = np.asarray(values[1:], dtype='float32')
        embedding_index[word] = coefs

# Create embedding matrix
embedding_matrix = np.zeros((vocab_size, embedding_dim))
for word, i in tokenizer.word_index.items():
    embedding_vector = embedding_index.get(word)
    if embedding_vector is not None:
        embedding_matrix[i] = embedding_vector
```

In [21]:
```
covered = [w for w in tokenizer.word_index if w in embedding_index]
print(f"Covered: {len(covered)} / {len(tokenizer.word_index)} words ({len(co
```

```
Covered: 11886 / 14195 words (83.7%)
```

In [22]:
```
y_train = train_df['target'].values
```

In [23]:
```
def build_model_with_glove():
    model = Sequential([
        Embedding(input_dim=vocab_size,
```

```
                     output_dim=embedding_dim,
                     weights=[embedding_matrix],
                     input_length=max_len,
                     trainable=True),  # fine-tune GloVe
        LSTM(64),
        Dropout(0.5),
        Dense(1, activation='sigmoid')
    ])
    model.compile(loss='binary_crossentropy', optimizer='adam',
                  metrics=['accuracy', tf.keras.metrics.AUC(name='auc')])
    return model
```

In [ ]:
```
early_stop = EarlyStopping(monitor='val_auc', patience=3, mode='max', restor

# Model B -fine-tuned GloVe
model_glove = build_model_with_glove()
history_glove = model_glove.fit(
    X_train_pad, y_train,
    validation_split=0.2,
    epochs=10,
    batch_size=32,
    callbacks=[early_stop],
    verbose=1
)
```

```
Epoch 1/10
2025-06-26 21:41:19.816154: W tensorflow/tsl/platform/profile_utils/cpu_util
s.cc:128] Failed to get CPU frequency: 0 Hz
191/191 [==============================] - 7s 32ms/step - loss: 0.6827 - acc
uracy: 0.5739 - auc: 0.5044 - val_loss: 0.6964 - val_accuracy: 0.5345 - val_
auc: 0.5000
Epoch 2/10
191/191 [==============================] - 6s 29ms/step - loss: 0.6815 - acc
uracy: 0.5793 - auc: 0.5037 - val_loss: 0.6928 - val_accuracy: 0.5345 - val_
auc: 0.5000
Epoch 3/10
191/191 [==============================] - 6s 29ms/step - loss: 0.6813 - acc
uracy: 0.5793 - auc: 0.4995 - val_loss: 0.6962 - val_accuracy: 0.5345 - val_
auc: 0.5000
Epoch 4/10
191/191 [==============================] - 6s 30ms/step - loss: 0.6812 - acc
uracy: 0.5793 - auc: 0.5019 - val_loss: 0.6919 - val_accuracy: 0.5345 - val_
auc: 0.5000
```

In [25]:
```
def build_model_from_scratch():
    model = Sequential([
        Embedding(input_dim=vocab_size,
                  output_dim=embedding_dim,
                  input_length=max_len,
                  trainable=True),  # random + trainable
        LSTM(64),
        Dropout(0.5),
        Dense(1, activation='sigmoid')
    ])
    model.compile(loss='binary_crossentropy', optimizer='adam',
```

```
                  metrics=['accuracy', tf.keras.metrics.AUC(name='auc')])
    return model
```

In [ ]:
```
# Model C - rainable embedding from scratch
model_scratch = build_model_from_scratch()
history_scratch = model_scratch.fit(
    X_train_pad, y_train,
    validation_split=0.2,
    epochs=10,
    batch_size=32,
    callbacks=[early_stop],
    verbose=1
)
```

```
Epoch 1/10
191/191 [==============================] - 6s 30ms/step - loss: 0.6826 - acc
uracy: 0.5750 - auc: 0.5014 - val_loss: 0.6963 - val_accuracy: 0.5345 - val_
auc: 0.5000
Epoch 2/10
191/191 [==============================] - 6s 30ms/step - loss: 0.6819 - acc
uracy: 0.5793 - auc: 0.4970 - val_loss: 0.6938 - val_accuracy: 0.5345 - val_
auc: 0.5000
Epoch 3/10
191/191 [==============================] - 6s 31ms/step - loss: 0.6811 - acc
uracy: 0.5793 - auc: 0.5088 - val_loss: 0.6930 - val_accuracy: 0.5345 - val_
auc: 0.5000
Epoch 4/10
191/191 [==============================] - 6s 30ms/step - loss: 0.6821 - acc
uracy: 0.5793 - auc: 0.4854 - val_loss: 0.6924 - val_accuracy: 0.5345 - val_
auc: 0.5000
```

In [ ]:
```
import numpy as np
print(f"% of completely empty sequences: {np.mean(np.sum(X_train_pad, axis=1
```

```
% of completely empty sequences: 0.04%
```

In [ ]:
```
for i in range(3):
    print("Original:", train_df['clean_text'].iloc[i])
    print("Sequence:", tokenizer.texts_to_sequences([train_df['clean_text'].
```

```
Original: deeds reason may allah forgive us
Sequence: [4084, 699, 54, 2562, 4085, 13]
Original: forest fire near la ronge sask canada
Sequence: [100, 4, 127, 558, 6060, 6061, 1406]
Original: residents asked shelter place notified officers evacuation shelter
place orders expected
Sequence: [1531, 1407, 1890, 538, 6062, 1532, 151, 1890, 538, 1202, 914]
```

In [ ]:
```
def build_simple_model():
    model = Sequential([
        Embedding(input_dim=vocab_size,
                  output_dim=embedding_dim,
                  input_length=max_len,
                  trainable=True),
        GlobalAveragePooling1D(),
        Dropout(0.3),
        Dense(1, activation='sigmoid')
```

```
    ])
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['ac
    return model

simple_model = build_simple_model()

history_simple = simple_model.fit(
    X_train_pad, y_train,
    validation_split=0.2,
    epochs=10,
    batch_size=32,
    callbacks=[early_stop],
    verbose=1
)
```

```
Epoch 1/10
191/191 [==============================] - 1s 4ms/step - loss: 0.6784 - accu
racy: 0.5796 - auc: 0.5547 - val_loss: 0.6913 - val_accuracy: 0.5345 - val_a
uc: 0.7798
Epoch 2/10
191/191 [==============================] - 1s 4ms/step - loss: 0.6669 - accu
racy: 0.5793 - auc: 0.6664 - val_loss: 0.6831 - val_accuracy: 0.5345 - val_a
uc: 0.8432
Epoch 3/10
191/191 [==============================] - 1s 4ms/step - loss: 0.6442 - accu
racy: 0.6051 - auc: 0.8225 - val_loss: 0.6616 - val_accuracy: 0.5522 - val_a
uc: 0.8475
Epoch 4/10
191/191 [==============================] - 1s 4ms/step - loss: 0.6067 - accu
racy: 0.6888 - auc: 0.8629 - val_loss: 0.6325 - val_accuracy: 0.6369 - val_a
uc: 0.8473
Epoch 5/10
191/191 [==============================] - 1s 4ms/step - loss: 0.5549 - accu
racy: 0.7654 - auc: 0.8999 - val_loss: 0.6017 - val_accuracy: 0.6927 - val_a
uc: 0.8467
Epoch 6/10
191/191 [==============================] - 1s 4ms/step - loss: 0.5016 - accu
racy: 0.8233 - auc: 0.9155 - val_loss: 0.5675 - val_accuracy: 0.7505 - val_a
uc: 0.8476
Epoch 7/10
191/191 [==============================] - 1s 4ms/step - loss: 0.4549 - accu
racy: 0.8417 - auc: 0.9227 - val_loss: 0.5438 - val_accuracy: 0.7820 - val_a
uc: 0.8473
Epoch 8/10
191/191 [==============================] - 1s 4ms/step - loss: 0.4151 - accu
racy: 0.8588 - auc: 0.9324 - val_loss: 0.5218 - val_accuracy: 0.7833 - val_a
uc: 0.8490
Epoch 9/10
191/191 [==============================] - 1s 3ms/step - loss: 0.3813 - accu
racy: 0.8714 - auc: 0.9390 - val_loss: 0.5052 - val_accuracy: 0.7965 - val_a
uc: 0.8490
Epoch 10/10
191/191 [==============================] - 1s 4ms/step - loss: 0.3525 - accu
racy: 0.8801 - auc: 0.9457 - val_loss: 0.4938 - val_accuracy: 0.7971 - val_a
uc: 0.8496
```

In [31]:

```
def build_simple_model_v2():
    model = Sequential([
        Embedding(input_dim=vocab_size,
                  output_dim=embedding_dim,
                  input_length=max_len,
                  trainable=True),
        GlobalAveragePooling1D(),
        Dropout(0.4),  # Increased dropout
        Dense(64, activation='relu'),  # New hidden layer
        Dropout(0.3),
        Dense(1, activation='sigmoid')
    ])
    model.compile(loss='binary_crossentropy',
                  optimizer='adam',
                  metrics=['accuracy', tf.keras.metrics.AUC(name='auc')])
    return model
```

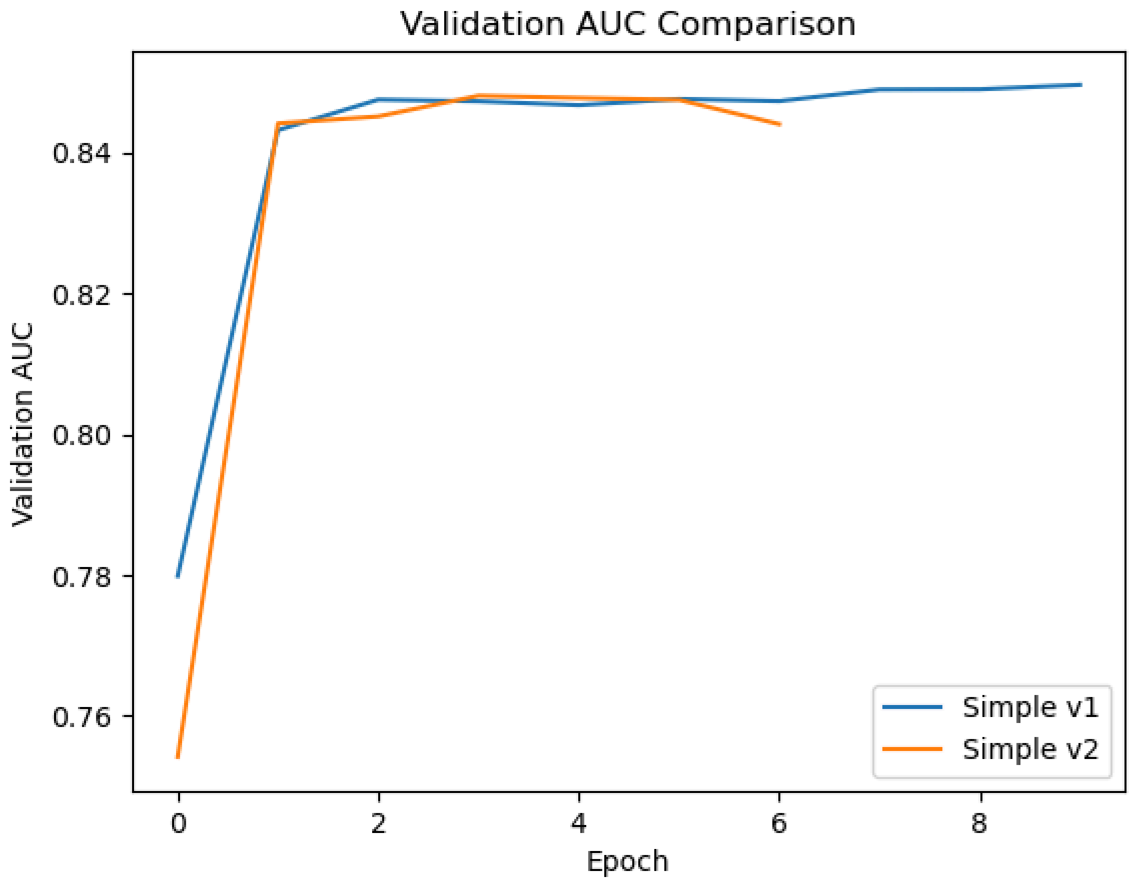
In [32]:

```
# Rebuild the model
model_simple_v2 = build_simple_model_v2()

# Use early stopping based on validation AUC
early_stop = EarlyStopping(monitor='val_auc', patience=3, mode='max', restor

# Train the model
history_simple_v2 = model_simple_v2.fit(
    X_train_pad, y_train,
    validation_split=0.2,
    epochs=10,
    batch_size=32,
    callbacks=[early_stop],
    verbose=1
)
```

```
Epoch 1/10
191/191 [==============================] - 1s 4ms/step - loss: 0.6797 - accu
racy: 0.5788 - auc: 0.5234 - val_loss: 0.6877 - val_accuracy: 0.5345 - val_a
uc: 0.7542
Epoch 2/10
191/191 [==============================] - 1s 4ms/step - loss: 0.6638 - accu
racy: 0.5869 - auc: 0.6498 - val_loss: 0.6683 - val_accuracy: 0.5364 - val_a
uc: 0.8441
Epoch 3/10
191/191 [==============================] - 1s 4ms/step - loss: 0.5449 - accu
racy: 0.7471 - auc: 0.8314 - val_loss: 0.5218 - val_accuracy: 0.7958 - val_a
uc: 0.8451
Epoch 4/10
191/191 [==============================] - 1s 4ms/step - loss: 0.3911 - accu
racy: 0.8401 - auc: 0.9054 - val_loss: 0.4755 - val_accuracy: 0.7984 - val_a
uc: 0.8481
Epoch 5/10
191/191 [==============================] - 1s 4ms/step - loss: 0.3163 - accu
racy: 0.8750 - auc: 0.9352 - val_loss: 0.4677 - val_accuracy: 0.7912 - val_a
uc: 0.8478
Epoch 6/10
191/191 [==============================] - 1s 4ms/step - loss: 0.2686 - accu
racy: 0.8947 - auc: 0.9529 - val_loss: 0.4955 - val_accuracy: 0.7853 - val_a
uc: 0.8475
Epoch 7/10
191/191 [==============================] - 1s 3ms/step - loss: 0.2312 - accu
racy: 0.9103 - auc: 0.9651 - val_loss: 0.4982 - val_accuracy: 0.7676 - val_a
uc: 0.8440
```

In [33]:

```python
# Plot AUC curves
plt.plot(history_simple.history['val_auc'], label='Simple v1')
plt.plot(history_simple_v2.history['val_auc'], label='Simple v2')
plt.xlabel('Epoch')
plt.ylabel('Validation AUC')
plt.title('Validation AUC Comparison')
plt.legend()
plt.show()

# Print AUC values numerically
print("Epoch\tSimple v1 AUC\tSimple v2 AUC")
for i in range(len(history_simple_v2.history['val_auc'])):
    auc_v1 = history_simple.history['val_auc'][i] if i < len(history_simple.
    auc_v2 = history_simple_v2.history['val_auc'][i]
    print(f"{i+1}\t{auc_v1:.4f}\t\t{auc_v2:.4f}")
```




```
Epoch   Simple v1 AUC   Simple v2 AUC
1       0.7798          0.7542
2       0.8432          0.8441
3       0.8475          0.8451
4       0.8473          0.8481
5       0.8467          0.8478
6       0.8476          0.8475
7       0.8473          0.8440
```

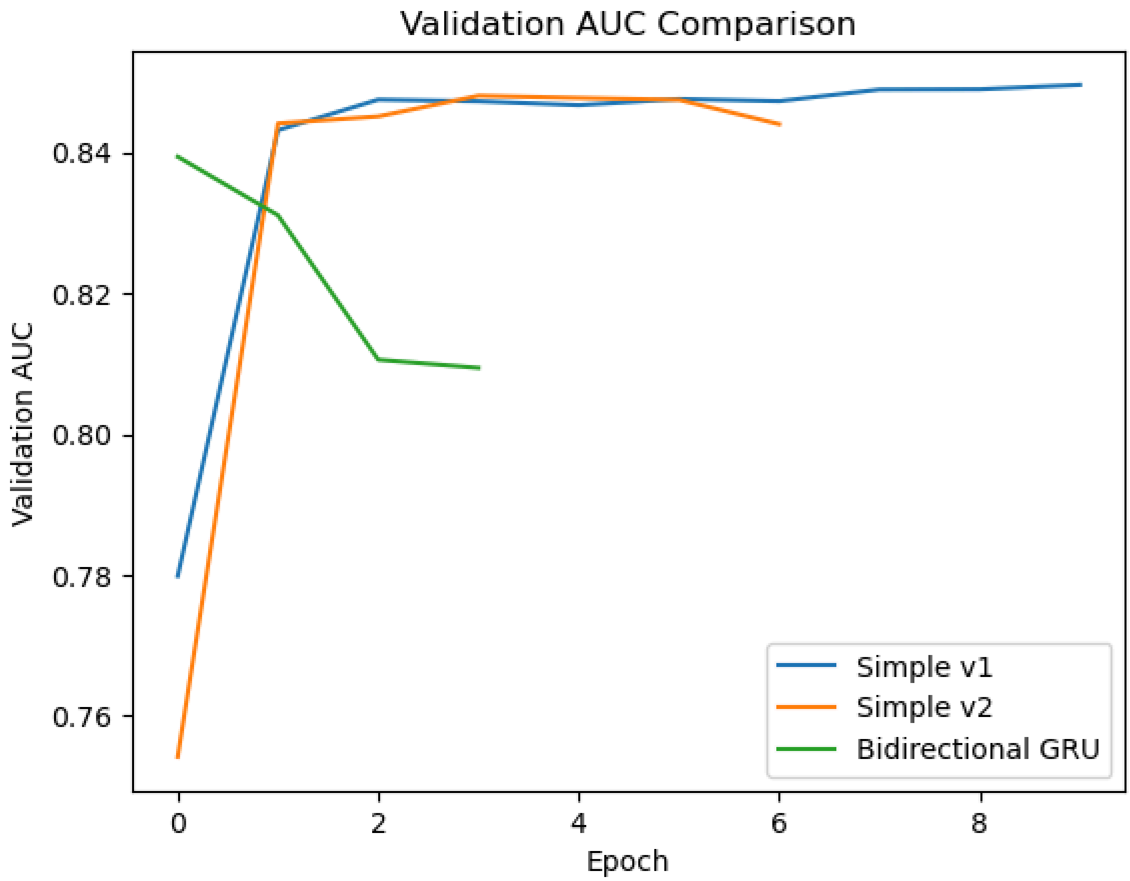
In [34]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, Bidirectional, GRU, Dense, Dr

def build_bidirectional_gru_model():
    model = Sequential([
        Embedding(input_dim=vocab_size,
                  output_dim=embedding_dim,
                  input_length=max_len,
                  trainable=True),
        Bidirectional(GRU(64, return_sequences=False)),
        Dropout(0.5),
        Dense(64, activation='relu'),
        Dropout(0.3),
        Dense(1, activation='sigmoid')
    ])
    model.compile(loss='binary_crossentropy',
                  optimizer='adam',
                  metrics=['accuracy', tf.keras.metrics.AUC(name='auc')])
    return model
```

```
# Initialize the model
model_bidir_gru = build_bidirectional_gru_model()

# Early stopping
early_stop = EarlyStopping(monitor='val_auc', patience=3, mode='max', restor

# Train
history_bidir_gru = model_bidir_gru.fit(
    X_train_pad, y_train,
    validation_split=0.2,
    epochs=10,
    batch_size=32,
    callbacks=[early_stop],
    verbose=1
)
```

```
Epoch 1/10
191/191 [==============================] - 6s 27ms/step - loss: 0.5556 - acc
uracy: 0.7174 - auc: 0.7678 - val_loss: 0.4925 - val_accuracy: 0.7748 - val_
auc: 0.8394
Epoch 2/10
191/191 [==============================] - 5s 25ms/step - loss: 0.3134 - acc
uracy: 0.8754 - auc: 0.9334 - val_loss: 0.5172 - val_accuracy: 0.7590 - val_
auc: 0.8311
Epoch 3/10
191/191 [==============================] - 5s 27ms/step - loss: 0.1843 - acc
uracy: 0.9365 - auc: 0.9756 - val_loss: 0.6167 - val_accuracy: 0.7387 - val_
auc: 0.8106
Epoch 4/10
191/191 [==============================] - 5s 28ms/step - loss: 0.1285 - acc
uracy: 0.9585 - auc: 0.9879 - val_loss: 0.6836 - val_accuracy: 0.7492 - val_
auc: 0.8094
```

In [ ]:

```
# Plot validation AUC
plt.plot(history_simple.history['val_auc'], label='Simple v1')
plt.plot(history_simple_v2.history['val_auc'], label='Simple v2')
plt.plot(history_bidir_gru.history['val_auc'], label='Bidirectional GRU')
plt.xlabel('Epoch')
plt.ylabel('Validation AUC')
plt.title('Validation AUC Comparison')
plt.legend()
plt.show()

# Print AUC values
print("Epoch\tSimple v1 AUC\tSimple v2 AUC\tBidirectional GRU AUC")
for i in range(len(history_bidir_gru.history['val_auc'])):
    auc_v1 = history_simple.history['val_auc'][i] if i < len(history_simple.
    auc_v2 = history_simple_v2.history['val_auc'][i] if i < len(history_simp
    auc_gru = history_bidir_gru.history['val_auc'][i]
    print(f"{i+1}\t{auc_v1:.4f}\t\t{auc_v2:.4f}\t\t{auc_gru:.4f}")
```




```
Epoch    Simple v1 AUC    Simple v2 AUC    Bidirectional GRU AUC
1        0.7798           0.7542           0.8394
2        0.8432           0.8441           0.8311
3        0.8475           0.8451           0.8106
4        0.8473           0.8481           0.8094
```

In [ ]:
```
def build_bidirectional_lstm_model():
    model = Sequential([
        Embedding(input_dim=vocab_size,
                  output_dim=embedding_dim,
                  input_length=max_len,
                  trainable=True),
        Bidirectional(LSTM(64, return_sequences=False)),
        Dropout(0.5),
        Dense(64, activation='relu'),
        Dropout(0.3),
        Dense(1, activation='sigmoid')
    ])
    model.compile(loss='binary_crossentropy',
                  optimizer='adam',
                  metrics=['accuracy', tf.keras.metrics.AUC(name='auc')])
    return model

model_bidir_lstm = build_bidirectional_lstm_model()

history_bidir_lstm = model_bidir_lstm.fit(
    X_train_pad, y_train,
    validation_split=0.2,
    epochs=10,
```

```
    batch_size=32,
    callbacks=[early_stop],
    verbose=1
)
```

```
Epoch 1/10
191/191 [==============================] - 8s 33ms/step - loss: 0.5712 - acc
uracy: 0.7000 - auc: 0.7534 - val_loss: 0.4742 - val_accuracy: 0.7879 - val_
auc: 0.8410
Epoch 2/10
191/191 [==============================] - 6s 29ms/step - loss: 0.3218 - acc
uracy: 0.8688 - auc: 0.9297 - val_loss: 0.5027 - val_accuracy: 0.7774 - val_
auc: 0.8310
Epoch 3/10
191/191 [==============================] - 6s 30ms/step - loss: 0.1903 - acc
uracy: 0.9335 - auc: 0.9732 - val_loss: 0.6046 - val_accuracy: 0.7308 - val_
auc: 0.7962
Epoch 4/10
191/191 [==============================] - 6s 29ms/step - loss: 0.1392 - acc
uracy: 0.9557 - auc: 0.9857 - val_loss: 0.7550 - val_accuracy: 0.7236 - val_
auc: 0.7883
```

In [ ]:

```
def build_bidirectional_lstm_with_glove():
    model = Sequential([
        Embedding(input_dim=vocab_size,
                  output_dim=embedding_dim,
                  weights=[embedding_matrix],
                  input_length=max_len,
                  trainable=False),  # Freeze GloVe
        Bidirectional(LSTM(64, return_sequences=False)),
        Dropout(0.5),
        Dense(64, activation='relu'),
        Dropout(0.3),
        Dense(1, activation='sigmoid')
    ])
    model.compile(loss='binary_crossentropy',
                  optimizer='adam',
                  metrics=['accuracy', tf.keras.metrics.AUC(name='auc')])
    return model

model_bidir_lstm_glove = build_bidirectional_lstm_with_glove()

history_bidir_lstm_glove = model_bidir_lstm_glove.fit(
    X_train_pad, y_train,
    validation_split=0.2,
    epochs=10,
    batch_size=32,
    callbacks=[early_stop],
    verbose=1
)
```

```
Epoch 1/10
191/191 [==============================] - 6s 24ms/step - loss: 0.5296 - acc
uracy: 0.7478 - auc: 0.8024 - val_loss: 0.4460 - val_accuracy: 0.7965 - val_
auc: 0.8660
Epoch 2/10
191/191 [==============================] - 4s 23ms/step - loss: 0.4555 - acc
uracy: 0.8000 - auc: 0.8537 - val_loss: 0.4632 - val_accuracy: 0.7892 - val_
auc: 0.8671
Epoch 3/10
191/191 [==============================] - 4s 23ms/step - loss: 0.4395 - acc
uracy: 0.8107 - auc: 0.8658 - val_loss: 0.4384 - val_accuracy: 0.8076 - val_
auc: 0.8690
Epoch 4/10
191/191 [==============================] - 5s 25ms/step - loss: 0.4256 - acc
uracy: 0.8128 - auc: 0.8732 - val_loss: 0.4415 - val_accuracy: 0.8017 - val_
auc: 0.8686
Epoch 5/10
191/191 [==============================] - 5s 24ms/step - loss: 0.4133 - acc
uracy: 0.8212 - auc: 0.8798 - val_loss: 0.4542 - val_accuracy: 0.7905 - val_
auc: 0.8650
Epoch 6/10
191/191 [==============================] - 4s 23ms/step - loss: 0.3946 - acc
uracy: 0.8312 - auc: 0.8916 - val_loss: 0.4601 - val_accuracy: 0.7965 - val_
auc: 0.8624
```

In [ ]:

```python
def build_bidirectional_lstm_glove_v2():
    model = Sequential([
        Embedding(input_dim=vocab_size,
                  output_dim=embedding_dim,
                  weights=[embedding_matrix],
                  input_length=max_len,
                  trainable=True),  # Fine-tuning GloVe
        Bidirectional(LSTM(32, return_sequences=False)),  # Smaller LSTM
        Dropout(0.5),
        Dense(64, activation='relu'),
        Dropout(0.3),
        Dense(1, activation='sigmoid')
    ])
    model.compile(loss='binary_crossentropy',
                  optimizer='adam',
                  metrics=['accuracy', tf.keras.metrics.AUC(name='auc')])
    return model

model_bidir_lstm_glove_v2 = build_bidirectional_lstm_glove_v2()

history_bidir_lstm_glove_v2 = model_bidir_lstm_glove_v2.fit(
    X_train_pad, y_train,
    validation_split=0.2,
    epochs=10,
    batch_size=32,
    callbacks=[early_stop],
    verbose=1
)
```

```
Epoch 1/10
191/191 [==============================] - 6s 22ms/step - loss: 0.5372 - acc
uracy: 0.7419 - auc: 0.7944 - val_loss: 0.4596 - val_accuracy: 0.7932 - val_
auc: 0.8619
Epoch 2/10
191/191 [==============================] - 4s 20ms/step - loss: 0.4173 - acc
uracy: 0.8279 - auc: 0.8770 - val_loss: 0.4370 - val_accuracy: 0.8076 - val_
auc: 0.8688
Epoch 3/10
191/191 [==============================] - 4s 20ms/step - loss: 0.3466 - acc
uracy: 0.8647 - auc: 0.9150 - val_loss: 0.4671 - val_accuracy: 0.7991 - val_
auc: 0.8667
Epoch 4/10
191/191 [==============================] - 4s 20ms/step - loss: 0.2737 - acc
uracy: 0.8970 - auc: 0.9469 - val_loss: 0.4917 - val_accuracy: 0.7997 - val_
auc: 0.8610
Epoch 5/10
191/191 [==============================] - 4s 20ms/step - loss: 0.2054 - acc
uracy: 0.9264 - auc: 0.9684 - val_loss: 0.6259 - val_accuracy: 0.7840 - val_
auc: 0.8495
```

In [ ]:

```
def build_bidirectional_lstm_glove_v3():
    model = Sequential([
        Embedding(input_dim=vocab_size,
                  output_dim=embedding_dim,
                  weights=[embedding_matrix],
                  input_length=max_len,
                  trainable=False),  # Frozen GloVe
        Bidirectional(LSTM(32, return_sequences=False)),  # Smaller LSTM
        Dropout(0.5),  # Strong regularization
        Dense(1, activation='sigmoid')  # Final classifier
    ])
    model.compile(loss='binary_crossentropy',
                  optimizer='adam',
                  metrics=['accuracy', tf.keras.metrics.AUC(name='auc')])
    return model

model_bidir_lstm_glove_v3 = build_bidirectional_lstm_glove_v3()

history_bidir_lstm_glove_v3 = model_bidir_lstm_glove_v3.fit(
    X_train_pad, y_train,
    validation_split=0.2,
    epochs=10,
    batch_size=32,
    callbacks=[early_stop],
    verbose=1
)
```

```
Epoch 1/10
191/191 [==============================] - 4s 15ms/step - loss: 0.5378 - acc
uracy: 0.7383 - auc: 0.7957 - val_loss: 0.4577 - val_accuracy: 0.7905 - val_
auc: 0.8552
Epoch 2/10
191/191 [==============================] - 3s 14ms/step - loss: 0.4668 - acc
uracy: 0.7911 - auc: 0.8456 - val_loss: 0.4454 - val_accuracy: 0.8017 - val_
auc: 0.8632
Epoch 3/10
191/191 [==============================] - 3s 14ms/step - loss: 0.4438 - acc
uracy: 0.8038 - auc: 0.8613 - val_loss: 0.4398 - val_accuracy: 0.8050 - val_
auc: 0.8672
Epoch 4/10
191/191 [==============================] - 3s 14ms/step - loss: 0.4344 - acc
uracy: 0.8102 - auc: 0.8650 - val_loss: 0.4423 - val_accuracy: 0.8011 - val_
auc: 0.8673
Epoch 5/10
191/191 [==============================] - 3s 14ms/step - loss: 0.4225 - acc
uracy: 0.8133 - auc: 0.8740 - val_loss: 0.4556 - val_accuracy: 0.7945 - val_
auc: 0.8655
Epoch 6/10
191/191 [==============================] - 2s 13ms/step - loss: 0.4093 - acc
uracy: 0.8256 - auc: 0.8813 - val_loss: 0.4563 - val_accuracy: 0.7951 - val_
auc: 0.8628
Epoch 7/10
191/191 [==============================] - 2s 13ms/step - loss: 0.3995 - acc
uracy: 0.8345 - auc: 0.8865 - val_loss: 0.4437 - val_accuracy: 0.8017 - val_
auc: 0.8679
Epoch 8/10
191/191 [==============================] - 3s 14ms/step - loss: 0.3902 - acc
uracy: 0.8373 - auc: 0.8909 - val_loss: 0.4467 - val_accuracy: 0.8063 - val_
auc: 0.8670
Epoch 9/10
191/191 [==============================] - 3s 14ms/step - loss: 0.3725 - acc
uracy: 0.8435 - auc: 0.9011 - val_loss: 0.4546 - val_accuracy: 0.8050 - val_
auc: 0.8643
Epoch 10/10
191/191 [==============================] - 3s 14ms/step - loss: 0.3652 - acc
uracy: 0.8458 - auc: 0.9043 - val_loss: 0.4997 - val_accuracy: 0.7958 - val_
auc: 0.8599
```

In [ ]:

```
# Combine keyword and text
train_df['combined_text'] = train_df['keyword'].fillna('') + ' ' + train_df[
test_df['combined_text'] = test_df['keyword'].fillna('') + ' ' + test_df['te

# Apply cleaning function
train_df['clean_combined_text'] = train_df['combined_text'].apply(clean_text
test_df['clean_combined_text'] = test_df['combined_text'].apply(clean_text)
```

In [ ]:

```
# Tokenizer
tokenizer = Tokenizer()
tokenizer.fit_on_texts(train_df['clean_combined_text'])

# Convert to sequences
X_train_seq = tokenizer.texts_to_sequences(train_df['clean_combined_text'])
X_test_seq = tokenizer.texts_to_sequences(test_df['clean_combined_text'])
```

```
# Pad sequences
X_train_pad = pad_sequences(X_train_seq, maxlen=max_len, padding='post')
X_test_pad = pad_sequences(X_test_seq, maxlen=max_len, padding='post')

# Recompute vocab size
vocab_size = len(tokenizer.word_index) + 1
```

In [ ]:

```
# embedding_index already loaded earlier
embedding_matrix = np.zeros((vocab_size, embedding_dim))
for word, i in tokenizer.word_index.items():
    embedding_vector = embedding_index.get(word)
    if embedding_vector is not None:
        embedding_matrix[i] = embedding_vector
```

In [43]:

```
def build_bidirectional_lstm_with_glove_combined():
    model = Sequential([
        Embedding(input_dim=vocab_size,
                  output_dim=embedding_dim,
                  weights=[embedding_matrix],
                  input_length=max_len,
                  trainable=False),  # frozen GloVe
        Bidirectional(LSTM(64, return_sequences=False)),
        Dropout(0.5),
        Dense(64, activation='relu'),
        Dropout(0.3),
        Dense(1, activation='sigmoid')
    ])
    model.compile(loss='binary_crossentropy',
                  optimizer='adam',
                  metrics=['accuracy', tf.keras.metrics.AUC(name='auc')])
    return model

model_combined = build_bidirectional_lstm_with_glove_combined()

history_combined = model_combined.fit(
    X_train_pad, y_train,
    validation_split=0.2,
    epochs=10,
    batch_size=32,
    callbacks=[early_stop],
    verbose=1
)
```

```
Epoch 1/10
191/191 [==============================] - 5s 23ms/step - loss: 0.5316 - acc
uracy: 0.7383 - auc: 0.7979 - val_loss: 0.4353 - val_accuracy: 0.8122 - val_
auc: 0.8723
Epoch 2/10
191/191 [==============================] - 4s 21ms/step - loss: 0.4615 - acc
uracy: 0.7982 - auc: 0.8492 - val_loss: 0.4335 - val_accuracy: 0.7997 - val_
auc: 0.8728
Epoch 3/10
191/191 [==============================] - 4s 21ms/step - loss: 0.4415 - acc
uracy: 0.8056 - auc: 0.8640 - val_loss: 0.4423 - val_accuracy: 0.8024 - val_
auc: 0.8700
Epoch 4/10
191/191 [==============================] - 4s 21ms/step - loss: 0.4193 - acc
uracy: 0.8164 - auc: 0.8770 - val_loss: 0.4580 - val_accuracy: 0.7840 - val_
auc: 0.8685
Epoch 5/10
191/191 [==============================] - 4s 23ms/step - loss: 0.4080 - acc
uracy: 0.8243 - auc: 0.8842 - val_loss: 0.4518 - val_accuracy: 0.8024 - val_
auc: 0.8702
```

In [ ]:

```python
def build_model_1():
    model = Sequential([
        Embedding(input_dim=vocab_size,
                  output_dim=embedding_dim,
                  weights=[embedding_matrix],
                  input_length=max_len,
                  trainable=True),  # fine-tuned GloVe
        Bidirectional(LSTM(32, return_sequences=False)),
        Dropout(0.5),
        Dense(64, activation='relu'),
        Dropout(0.3),
        Dense(1, activation='sigmoid')
    ])
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['ac
    return model

model_1 = build_model_1()
history_1 = model_1.fit(X_train_pad, y_train, validation_split=0.2, epochs=1
```

```
Epoch 1/10
191/191 [==============================] - 5s 20ms/step - loss: 0.5533 - acc
uracy: 0.7204 - auc: 0.7809 - val_loss: 0.4477 - val_accuracy: 0.8043 - val_
auc: 0.8629
Epoch 2/10
191/191 [==============================] - 4s 19ms/step - loss: 0.4257 - acc
uracy: 0.8202 - auc: 0.8751 - val_loss: 0.4342 - val_accuracy: 0.8063 - val_
auc: 0.8707
Epoch 3/10
191/191 [==============================] - 4s 21ms/step - loss: 0.3510 - acc
uracy: 0.8611 - auc: 0.9126 - val_loss: 0.4428 - val_accuracy: 0.8011 - val_
auc: 0.8682
Epoch 4/10
191/191 [==============================] - 4s 22ms/step - loss: 0.2808 - acc
uracy: 0.8923 - auc: 0.9444 - val_loss: 0.5068 - val_accuracy: 0.7912 - val_
auc: 0.8553
Epoch 5/10
191/191 [==============================] - 3s 18ms/step - loss: 0.2098 - acc
uracy: 0.9243 - auc: 0.9681 - val_loss: 0.5905 - val_accuracy: 0.7846 - val_
auc: 0.8425
```

In [ ]:

```python
def build_model_2():
    model = Sequential([
        Embedding(input_dim=vocab_size,
                  output_dim=embedding_dim,
                  weights=[embedding_matrix],
                  input_length=max_len,
                  trainable=False),
        Bidirectional(LSTM(64, return_sequences=True)),  # return full seque
        GlobalMaxPooling1D(),  # pool over time dimension
        Dropout(0.4),
        Dense(1, activation='sigmoid')
    ])
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['ac
    return model

model_2 = build_model_2()
history_2 = model_2.fit(X_train_pad, y_train, validation_split=0.2, epochs=1
```

```
Epoch 1/10
191/191 [==============================] - 6s 25ms/step - loss: 0.5124 - acc
uracy: 0.7516 - auc: 0.8116 - val_loss: 0.4392 - val_accuracy: 0.8037 - val_
auc: 0.8672
Epoch 2/10
191/191 [==============================] - 5s 24ms/step - loss: 0.4451 - acc
uracy: 0.8036 - auc: 0.8607 - val_loss: 0.4305 - val_accuracy: 0.8043 - val_
auc: 0.8757
Epoch 3/10
191/191 [==============================] - 4s 23ms/step - loss: 0.4223 - acc
uracy: 0.8156 - auc: 0.8752 - val_loss: 0.4376 - val_accuracy: 0.8043 - val_
auc: 0.8735
Epoch 4/10
191/191 [==============================] - 4s 23ms/step - loss: 0.4079 - acc
uracy: 0.8266 - auc: 0.8847 - val_loss: 0.4221 - val_accuracy: 0.8089 - val_
auc: 0.8781
Epoch 5/10
191/191 [==============================] - 4s 23ms/step - loss: 0.3939 - acc
uracy: 0.8310 - auc: 0.8916 - val_loss: 0.4340 - val_accuracy: 0.8004 - val_
auc: 0.8751
Epoch 6/10
191/191 [==============================] - 5s 24ms/step - loss: 0.3715 - acc
uracy: 0.8414 - auc: 0.9056 - val_loss: 0.4653 - val_accuracy: 0.7879 - val_
auc: 0.8722
Epoch 7/10
191/191 [==============================] - 4s 23ms/step - loss: 0.3549 - acc
uracy: 0.8489 - auc: 0.9123 - val_loss: 0.4503 - val_accuracy: 0.7958 - val_
auc: 0.8701
```

In [ ]:

```python
def build_model_3():
    model = Sequential([
        Embedding(input_dim=vocab_size,
                  output_dim=embedding_dim,
                  weights=[embedding_matrix],
                  input_length=max_len,
                  trainable=False),
        Conv1D(filters=128, kernel_size=3, activation='relu'),
        GlobalMaxPooling1D(),
        Dropout(0.5),
        Dense(64, activation='relu'),
        Dropout(0.3),
        Dense(1, activation='sigmoid')
    ])
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['ac
    return model

model_3 = build_model_3()
history_3 = model_3.fit(X_train_pad, y_train, validation_split=0.2, epochs=1
```

```
Epoch 1/10
191/191 [==============================] - 1s 3ms/step - loss: 0.5999 - accu
racy: 0.6933 - auc: 0.7325 - val_loss: 0.4910 - val_accuracy: 0.7820 - val_a
uc: 0.8555
Epoch 2/10
191/191 [==============================] - 1s 3ms/step - loss: 0.4976 - accu
racy: 0.7658 - auc: 0.8285 - val_loss: 0.4634 - val_accuracy: 0.7945 - val_a
uc: 0.8641
Epoch 3/10
191/191 [==============================] - 1s 3ms/step - loss: 0.4665 - accu
racy: 0.7862 - auc: 0.8495 - val_loss: 0.4488 - val_accuracy: 0.8017 - val_a
uc: 0.8662
Epoch 4/10
191/191 [==============================] - 1s 3ms/step - loss: 0.4388 - accu
racy: 0.8044 - auc: 0.8665 - val_loss: 0.4501 - val_accuracy: 0.8037 - val_a
uc: 0.8683
Epoch 5/10
191/191 [==============================] - 1s 3ms/step - loss: 0.4113 - accu
racy: 0.8143 - auc: 0.8838 - val_loss: 0.4277 - val_accuracy: 0.7991 - val_a
uc: 0.8754
Epoch 6/10
191/191 [==============================] - 1s 3ms/step - loss: 0.3967 - accu
racy: 0.8282 - auc: 0.8944 - val_loss: 0.4317 - val_accuracy: 0.8030 - val_a
uc: 0.8744
Epoch 7/10
191/191 [==============================] - 1s 3ms/step - loss: 0.3755 - accu
racy: 0.8360 - auc: 0.9041 - val_loss: 0.4358 - val_accuracy: 0.8004 - val_a
uc: 0.8715
Epoch 8/10
191/191 [==============================] - 1s 3ms/step - loss: 0.3543 - accu
racy: 0.8479 - auc: 0.9139 - val_loss: 0.4417 - val_accuracy: 0.7991 - val_a
uc: 0.8702
```

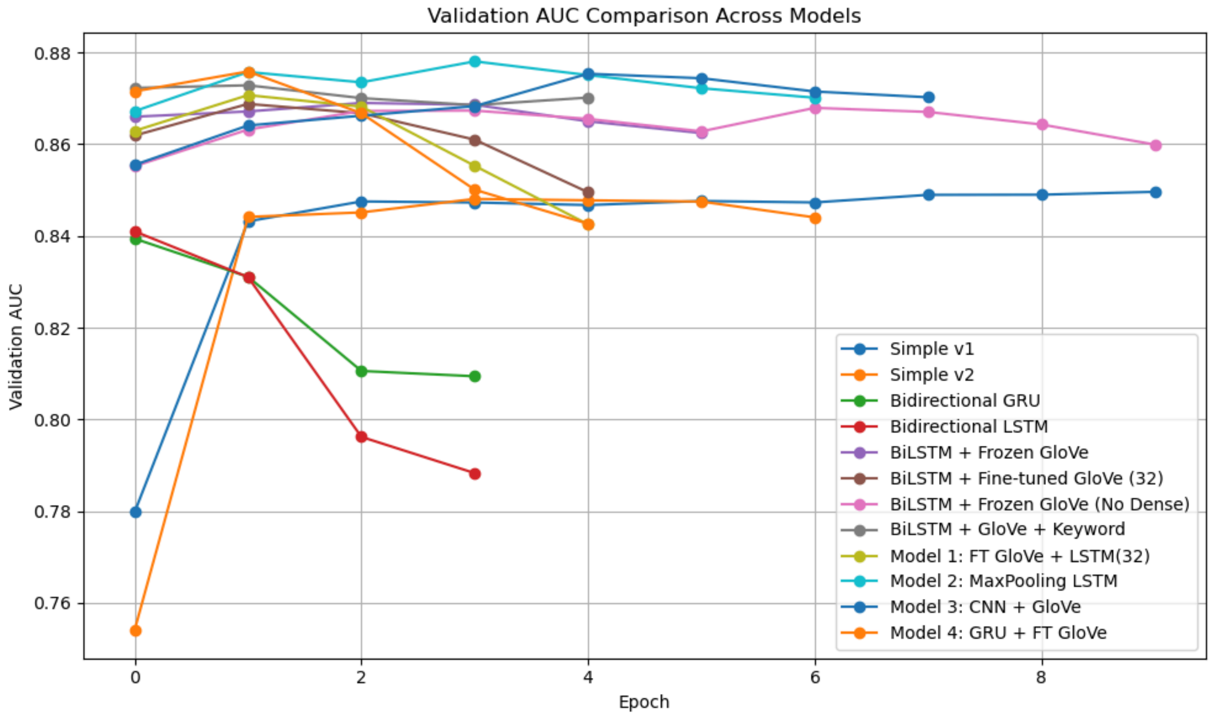
In [47]:

```
def build_model_4():
    model = Sequential([
        Embedding(input_dim=vocab_size,
                  output_dim=embedding_dim,
                  weights=[embedding_matrix],
                  input_length=max_len,
                  trainable=True),  # fine-tune GloVe
        Bidirectional(GRU(64)),
        Dropout(0.5),
        Dense(64, activation='relu'),
        Dropout(0.3),
        Dense(1, activation='sigmoid')
    ])
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['ac
    return model

model_4 = build_model_4()
history_4 = model_4.fit(X_train_pad, y_train, validation_split=0.2, epochs=1
```

```
Epoch 1/10
191/191 [==============================] - 6s 26ms/step - loss: 0.5508 - acc
uracy: 0.7251 - auc: 0.7800 - val_loss: 0.4598 - val_accuracy: 0.7899 - val_
auc: 0.8715
Epoch 2/10
191/191 [==============================] - 5s 25ms/step - loss: 0.4167 - acc
uracy: 0.8241 - auc: 0.8792 - val_loss: 0.4329 - val_accuracy: 0.8089 - val_
auc: 0.8759
Epoch 3/10
191/191 [==============================] - 5s 24ms/step - loss: 0.3364 - acc
uracy: 0.8647 - auc: 0.9221 - val_loss: 0.4622 - val_accuracy: 0.7965 - val_
auc: 0.8668
Epoch 4/10
191/191 [==============================] - 5s 25ms/step - loss: 0.2633 - acc
uracy: 0.8974 - auc: 0.9517 - val_loss: 0.5095 - val_accuracy: 0.7761 - val_
auc: 0.8501
Epoch 5/10
191/191 [==============================] - 6s 32ms/step - loss: 0.1871 - acc
uracy: 0.9325 - auc: 0.9744 - val_loss: 0.7012 - val_accuracy: 0.7623 - val_
auc: 0.8426
```

In [ ]:

```python
plt.figure(figsize=(10, 6))

# Plot validation AUCs.history['val_auc'], label='Simple v1', marker='o')
plt.plot(history_simple_v2.history['val_auc'], label='Simple v2', marker='o'
plt.plot(history_bidir_gru.history['val_auc'], label='Bidirectional GRU', ma
plt.plot(history_bidir_lstm.history['val_auc'], label='Bidirectional LSTM',
plt.plot(history_bidir_lstm_glove.history['val_auc'], label='BiLSTM + Frozer
plt.plot(history_bidir_lstm_glove_v2.history['val_auc'], label='BiLSTM + Fir
plt.plot(history_bidir_lstm_glove_v3.history['val_auc'], label='BiLSTM + Frc
plt.plot(history_combined.history['val_auc'], label='BiLSTM + GloVe + Keywor

# Add new models if they run
try: plt.plot(history_1.history['val_auc'], label='Model 1: FT GloVe + LSTM(
except: pass
try: plt.plot(history_2.history['val_auc'], label='Model 2: MaxPooling LSTM'
except: pass
try: plt.plot(history_3.history['val_auc'], label='Model 3: CNN + GloVe', ma
except: pass
try: plt.plot(history_4.history['val_auc'], label='Model 4: GRU + FT GloVe',
except: pass

plt.xlabel('Epoch')
plt.ylabel('Validation AUC')
plt.title('Validation AUC Comparison Across Models')
plt.legend(loc='lower right')
plt.grid(True)
plt.tight_layout()
plt.show()
```


Validation AUC Comparison Across Models
Validation AUC
Epoch
Simple v1
Simple v2
Bidirectional GRU
Bidirectional LSTM
BiLSTM + Frozen GloVe
BiLSTM + Fine-tuned GloVe (32)
BiLSTM + Frozen GloVe (No Dense)
BiLSTM + GloVe + Keyword
Model 1: FT GloVe + LSTM(32)
Model 2: MaxPooling LSTM
Model 3: CNN + GloVe
Model 4: GRU + FT GloVe