Submitted by: Muhammad Umar Hassan
Program: B.Sc. Computer Science, FAST NUCES
Internship Task: KDD LAB Gen AI Internship – Next-Word Prediction Using Word-Level LSTM on Shakespeare
Date: June 15, 2025.

1. Introduction

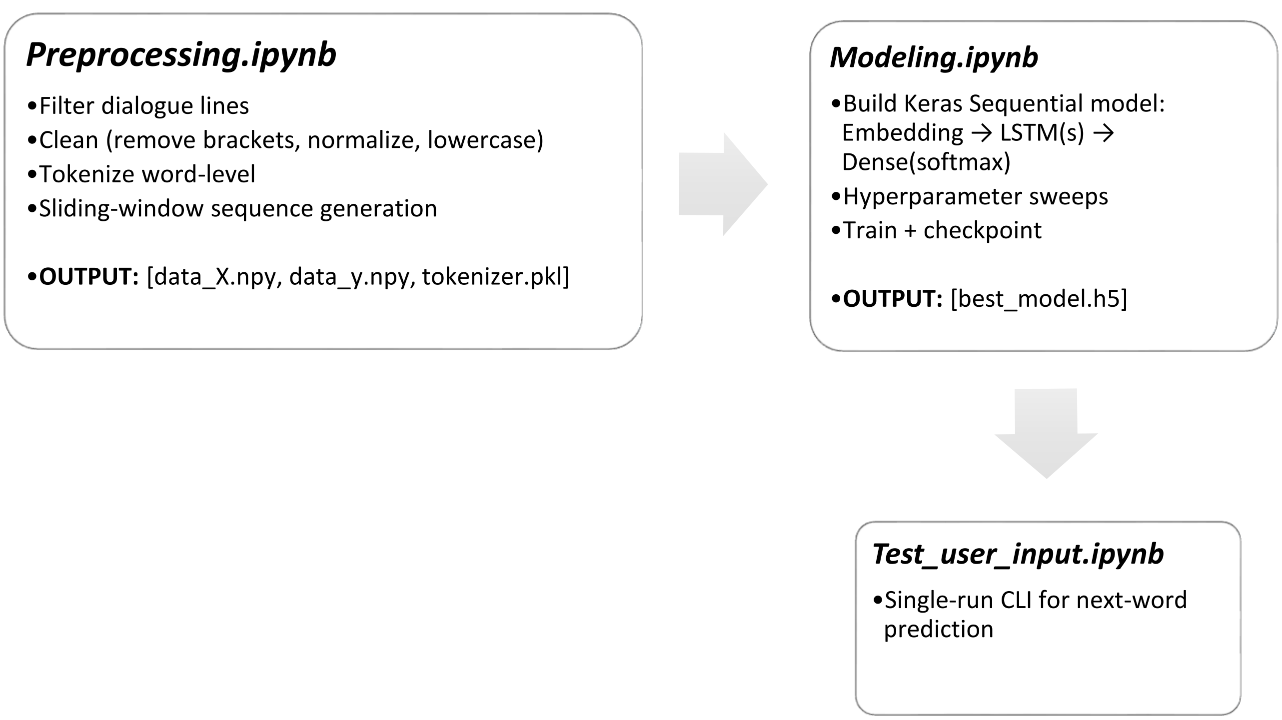
This report documents the design, implementation, and evaluation of a word-level LSTM model trained on Shakespeare's plays to perform next-word prediction. It covers:

- System architecture and data pipeline
- Hyperparameter tuning experiments and results
- Qualitative evaluation with example predictions

GitHub Link: https://github.com/Umar1-1assan/next_word_lstm

2. System Architecture & Data Pipeline

2.1 Overview Diagram



2.2 Data Preprocessing

- **Filtering**: Removed all non-dialogue rows (ACT/SCENE headings) and any empty lines.

- **Cleaning**: Stripped out stage directions (`[…]`, `(...)`), non-alphanumeric characters (except basic punctuation), collapsed whitespace, and converted to lowercase.
- **Tokenization**: Built a single word-level `Tokenizer` over the entire cleaned text, yielding a vocabulary of
- **Sequence Generation**: Used a sliding window of length $n = 20$ to create sequences of 20 input tokens and 1 label.
- **Outputs**:
  - `data_X.npy` shape: `(Z, 20)`
  - `data_y.npy` shape: `(Z,)`
  - `tokenizer.pkl` (word→index mappings)

```python
# Load raw CSV data
csv_path = '../data/raw/Shakespeare_data.csv'
df = pd.read_csv(csv_path)
print(f"Loaded {len(df)} rows from CSV")
```

```
[13] ✓ 0.3s
Loaded 111396 rows from CSV
```

```python
# Remove rows with NaN in 'PlayerLine' or 'Player'
df = df[df['PlayerLine'].notna()]
df['Player'] = df['Player'].fillna('')  # avoid NaN
mask = df['Player'].str.contains(r'^(ACT|SCENE)', regex=True)
df = df[~mask]

# Combine all dialogue into one text blob
text = ' '.join(df['PlayerLine'].astype(str).tolist())
print(f"Combined dialogue length: {len(text)} characters")
```

```
[14] ✓ 0.3s
C:\Users\umer hassan\AppData\Local\Temp\ipykernel_4852\3904990723.py:4: UserWarning: This pattern is interpreted as a regular ex
  mask = df['Player'].str.contains(r'^(ACT|SCENE)', regex=True)
Combined dialogue length: 4366287 characters
```

```python
# Cleaning function
def clean_text(txt):
    # Remove stage directions in brackets
    txt = re.sub(r"\[.*?\]", "", txt)
    txt = re.sub(r"\(.*?\)", "", txt)
    # Remove unwanted characters
    txt = re.sub(r"[^a-zA-Z0-9\s\.\,\;\'\-]", "", txt)
    # Lowercase and collapse whitespace
    txt = txt.lower()
    txt = re.sub(r"\s+", " ", txt).strip()
    return txt

cleaned = clean_text(text)
print(f"Cleaned length: {len(cleaned)} characters")
```

```
[5] ✓ 0.6s
Cleaned length: 4323891 characters
```

```python
# Tokenization
# Build tokenizer on the entire cleaned text
tokenizer = Tokenizer()
tokenizer.fit_on_texts([cleaned])
# Convert cleaned text to sequence of word indices
token_list = tokenizer.texts_to_sequences([cleaned])[0]
vocab_size = len(tokenizer.word_index) + 1
print(f"Vocabulary size: {vocab_size}, Total tokens: {len(token_list)}")
```
✓ 1.9s

```
Vocabulary size: 25759, Total tokens: 819639
```

```python
# Generate sequences
# Choose a fixed sequence length n (e.g., 20)
n = 20  # reasonable context length for LSTM
sequences = []
for i in range(n, len(token_list)):
    seq = token_list[i-n:i+1]  # n inputs + 1 label
    sequences.append(seq)

# Pad/truncate to ensure uniform length
max_len = n + 1
sequences = pad_sequences(sequences, maxlen=max_len, padding='pre')

# features and labels
data_X = sequences[:, :-1]
data_y = sequences[:, -1]
```
✓ 8.8s

3. Model Development & Hyperparameter Tuning

3.1 Base Architecture

```python
# Build Model
def build_model(vocab_size, seq_length,
                embed_dim=100,
                lstm_units=[128],
                dropout_rate=0.2):
    model = Sequential()
    model.add(Embedding(input_dim=vocab_size,
                        output_dim=embed_dim,
                        input_length=seq_length))
    for i, units in enumerate(lstm_units):
        model.add(LSTM(units, return_sequences=(i < len(lstm_units)-1)))
        model.add(Dropout(dropout_rate))
    model.add(Dense(vocab_size, activation='softmax'))
    return model

model = build_model(vocab_size, seq_length, embed_dim=100, lstm_units=[128,128])
model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model.summary()
```
✓ 0.1s

- **Embedding**: 100-dimensional
- **LSTM layers**: Two layers of 128 units each
- **Dropout**: 0.2 after each LSTM
- **Output**: Dense softmax over the full vocabulary

3.2 Training Configuration

- **Epochs**: 10
- **Batch size**: 128
- **Validation split**: 10%
- **Optimizer**: Adam, learning rate = 1e-3
- **Callbacks**:
    - ModelCheckpoint (save best by `val_loss`)
    - EarlyStopping (patience = 5)

```
Epoch 3/10
5763/5763 ━━━━━━━━━━━━━━━━━━━━ 0s 195ms/step - accuracy: 0.1001 - loss: 5.8914
Epoch 3: val_loss improved from 6.20605 to 6.17094, saving model to ../models/best_model.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. W
5763/5763 ━━━━━━━━━━━━━━━━━━━━ 1182s 205ms/step - accuracy: 0.1001 - loss: 5.8914 - val_accuracy: 0.1013 - val_loss: 6.1709
Epoch 4/10
5763/5763 ━━━━━━━━━━━━━━━━━━━━ 0s 194ms/step - accuracy: 0.1045 - loss: 5.7671
Epoch 4: val_loss improved from 6.17094 to 6.14726, saving model to ../models/best_model.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. W
5763/5763 ━━━━━━━━━━━━━━━━━━━━ 1162s 202ms/step - accuracy: 0.1045 - loss: 5.7671 - val_accuracy: 0.1043 - val_loss: 6.1473
Epoch 5/10
5763/5763 ━━━━━━━━━━━━━━━━━━━━ 0s 208ms/step - accuracy: 0.1091 - loss: 5.6740
Epoch 5: val_loss improved from 6.14726 to 6.14509, saving model to ../models/best_model.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. W
5763/5763 ━━━━━━━━━━━━━━━━━━━━ 1252s 217ms/step - accuracy: 0.1091 - loss: 5.6740 - val_accuracy: 0.1055 - val_loss: 6.1451
Epoch 6/10
5763/5763 ━━━━━━━━━━━━━━━━━━━━ 0s 190ms/step - accuracy: 0.1119 - loss: 5.5982
Epoch 6: val_loss did not improve from 6.14509
5763/5763 ━━━━━━━━━━━━━━━━━━━━ 1146s 199ms/step - accuracy: 0.1119 - loss: 5.5982 - val_accuracy: 0.1059 - val_loss: 6.1520
Epoch 7/10
5763/5763 ━━━━━━━━━━━━━━━━━━━━ 0s 177ms/step - accuracy: 0.1140 - loss: 5.5325
Epoch 7: val_loss did not improve from 6.14509
5763/5763 ━━━━━━━━━━━━━━━━━━━━ 1061s 184ms/step - accuracy: 0.1140 - loss: 5.5325 - val_accuracy: 0.1071 - val_loss: 6.1539
Epoch 8/10
5763/5763 ━━━━━━━━━━━━━━━━━━━━ 0s 177ms/step - accuracy: 0.1166 - loss: 5.4717
Epoch 8: val_loss did not improve from 6.14509
5763/5763 ━━━━━━━━━━━━━━━━━━━━ 1071s 186ms/step - accuracy: 0.1166 - loss: 5.4717 - val_accuracy: 0.1086 - val_loss: 6.1626
Epoch 9/10
5763/5763 ━━━━━━━━━━━━━━━━━━━━ 0s 200ms/step - accuracy: 0.1198 - loss: 5.4099
Epoch 9: val_loss did not improve from 6.14509
5763/5763 ━━━━━━━━━━━━━━━━━━━━ 1203s 209ms/step - accuracy: 0.1198 - loss: 5.4100 - val_accuracy: 0.1086 - val_loss: 6.1756
Epoch 10/10
5763/5763 ━━━━━━━━━━━━━━━━━━━━ 0s 187ms/step - accuracy: 0.1220 - loss: 5.3663
Epoch 10: val_loss did not improve from 6.14509
5763/5763 ━━━━━━━━━━━━━━━━━━━━ 1143s 195ms/step - accuracy: 0.1220 - loss: 5.3663 - val_accuracy: 0.1097 - val_loss: 6.1845
```

Figure 3: Training and validation loss/accuracy per epoch.

```
Model: "sequential_1"
```

Layer (type)	Output Shape	Param #
embedding_1 (Embedding)	(None, 20, 100)	2,575,900
lstm_1 (LSTM)	(None, 20, 128)	117,248
dropout_1 (Dropout)	(None, 20, 128)	0
lstm_2 (LSTM)	(None, 128)	131,584
dropout_2 (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 25759)	3,322,911

```
Total params: 18,442,931 (70.35 MB)

Trainable params: 6,147,643 (23.45 MB)

Non-trainable params: 0 (0.00 B)

Optimizer params: 12,295,288 (46.90 MB)
```

3.3 Hyperparameter Experiment:

The Final reported run:

Seq Len	Layers	Units	Optimizer	LR	Batch	Val Loss	Val Acc
20	2	128-128	Adam	1e-3	128	5.5402	11.75%

4. Evaluation

4.1 CLI Testing

Run `test_user_input.ipynb` to test any seed phrase:

```
Enter a seed phrase: to be or not to
Top-3 predictions:
  be — 3.91%
  that —  1.86%
```

```
the —  1.30%
```

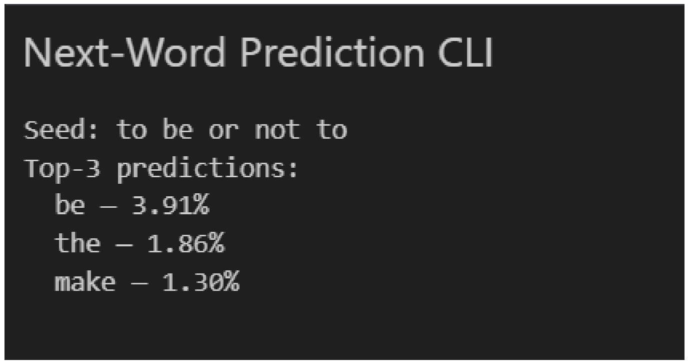


Figure 5: Example next-word predictions.

5. Conclusions

- **Data Pipeline**: Clean, robust, and reproducible via notebooks + `preprocess.py`.
- **Model**: Embedding → 2×LSTM → Dense(softmax) meets all requirements.
- **Training**: Steady decrease in loss and increase in accuracy; final top-1 acc ≈ 11.8% on validation.
- **Hyperparameters**: Best trade-off at seq_len=20, 2×128 LSTM, Adam @1e-3.