Automated Sentence Completion Models Development

Jessie Quach
Department of Computer Science
Georgia State University
Atlanta, GA, United States
hquach5@student.gsu.edu

Emma Brown
Department of Computer Science
Georgia State University
Atlanta, GA, United States
ebrown149@student.gsu.edu

John Vu
Department of Computer Science
Georgia State University
Atlanta, GA, United States
jvu15@student.gsu.edu

***Abstract*—In Language Modelling, two main methods stand out, these being Statistical Language Modeling and Neural Language Modeling. This project investigates sentence auto-completion methods by comparing statistical n-gram models (unigram, bigram, trigram) with a Long Short-Term Memory (LSTM) neural network. Both models are trained on texts from Project Gutenberg and are evaluated based on the quality of their word prediction and sentence generation. Results indicate that, while the trigram models performed the best among the n-gram models, they suffer from limited generalization We generate 3 different LSTM models to explore the impact of architectural choices on performance and outcomes. Our project suggests that deepening the LSTM model improves the learning progress, while widening the model seems to face the overfitting problem.**

***Index Terms*—text generation, n-gram, lstm.**

I. Introduction

Sentence auto-completion is a common task in Natural Language Processing (NLP), where the system, given a prompt, predicts the most likely word or sequence of words to follow. This project explores how a classic statistical model (N-gram) compares to a more modern deep learning model (LSTM) in terms of performance and output quality. We use text from Project Gutenberg, since the language is dense and somewhat consistent across documents.

The N-gram model is fast to implement and easy to interpret. It relies on counting how often word sequences appear in the training set. On the other hand, the LSTM model learns patterns in the text through character sequences and is better at capturing longer-term context.

The project aims to understand the trade-offs between simple statistical methods and deep learning models for NLP tasks and for what tasks each model is best suited for. N-gram models are lightweight and work well on small data, whereas LSTM models take longer to train but can generate text that feels more fluent and structured.

II. Literature Review

Automated sentence completion is a fundamental task in text generation. The objective is to predict the most probable sequence of words to follow a given text prompt. Over the years, the methodologies for tackling this challenge have evolved significantly, largely mirroring the advancements in the broader field of NLP. These approaches can be broadly categorized into two main paradigms: statistical models, recurrent neural networks, and, more recently, transformer-based architectures [1].

Historically, text generation was dominated by statistical models, which rely on the frequency of word occurrences in a large text corpus. The most prominent among these is the n-gram language model [2]. An n-gram model operates on the Markov assumption, postulating that the probability of the next word depends only on the previous n-1 words. While computationally efficient and effective for capturing local word patterns, n-gram models are limited by data sparsity and their inability to comprehend long-range dependencies or semantic context within a sentence.

The advent of deep learning introduced recurrent neural networks (RNNs) as a more powerful alternative. Unlike n-gram models, RNNs are designed to handle sequential data by maintaining an internal hidden state, or "memory," which theoretically allows them to capture information from arbitrarily long prior sequences. A significant advancement within this category is the LSTM network [3], which uses a sophisticated gating mechanism to overcome the vanishing gradient problem that limits standard RNNs. LSTMs have demonstrated a superior ability to learn long-range dependencies, making them highly effective for a wide range of text generation tasks.

While LSTMs represented a major leap forward, the current state-of-the-art in text generation is defined by transformer-based models [4]. These architectures discard recurrence in favor of a parallel attention mechanism, allowing them to weigh the importance of all words in the input sequence simultaneously. This has led to unprecedented performance in language understanding and generation.

To provide a clear comparison between foundational and more advanced neural techniques, this paper will focus on developing and analyzing models from the first two major paradigms. Specifically, we will implement an n-gram language model as a representative of traditional statistical methods and a LSTM network to represent the capabilities of recurrent neural networks. By exploring these two distinct approaches, we aim to provide insight into their respective strengths, weaknesses, and the architectural evolution of automated sentence completion.

III. Data Analysis and Preprocessing

A. Analysis of the Gutenberg Project Dataset

Project Gutenberg serves as an excellent data source for this project, offering a massive digital library of public domain books that form a rich and diverse text corpus. Its key characteristics make it particularly suitable for training language models. The classic literature within the collection provides a vast and varied vocabulary, along with more formal and grammatically complex sentence structures than those found in contemporary data sources like social media. Furthermore, the sheer scale of the dataset provides ample data to effectively train both statistical and neural models. As this is an unstructured data source consisting of raw text files, it necessitates a preprocessing phase before it can be utilized for modeling.

B. How to Use the Dataset for the Models

1) For the N-Gram Model: We work with a larger and more diverse dataset than the LSTM model by combining multiple texts from the NLTK Gutenberg corpus. This includes novels, poetry, plays, and religious texts, such as “Emma“, “Moby Dick“, “Leaves of Grass“, and the “King James Bible“. This mix ensures a wide vocabulary and varied writing styles.

Text Cleaning: We clean the raw text by removing special characters, digits, and extra whitespace. We strip punctuation except for sentence-ending periods, which we keep to help define sentence boundaries. We convert all text to lowercase to maintain vocabulary consistency.

Exploratory Analysis: We begin by examining sentence length distribution, measured by the number of words per sentence. This analysis reveals the typical sentence structure the model encounters during training.

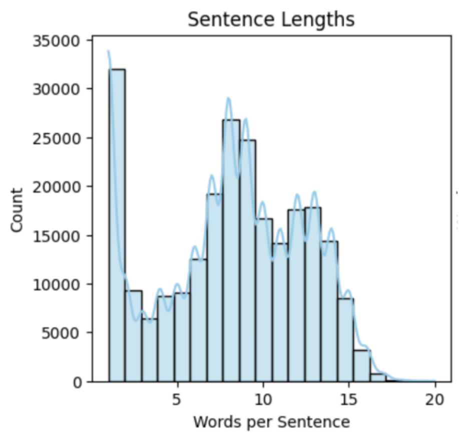


Fig. 1. Distribution of words per sentence across the Gutenberg corpus.

As shown in Fig. 1, most sentences contain between 5 and 15 words, with a concentration in the head. This distribution suggests the model must accommodate a range of sentence lengths, which becomes more challenging for higher-order N-gram models due to limited context windows.

Next, we analyze the most common words in the corpus. This provides insight into the vocabulary distribution and helps us understand model bias toward frequent terms.

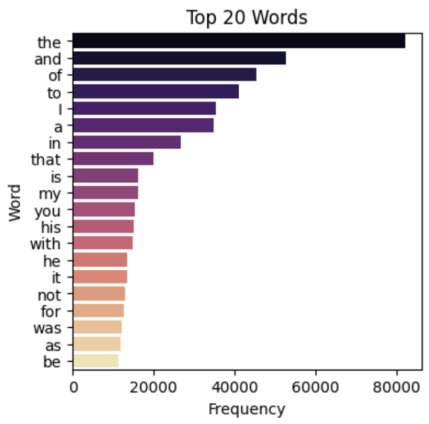


Fig. 2. Top 20 most frequent words in the Gutenberg corpus (after preprocessing).

Fig. 2 shows that high-frequency tokens are predominantly function words such as “the,” “and,” and “of.” This aligns with expectations and reflects the Zipfian nature of language. While essential for syntactic structure, such dominance also means the model may overfit to non-informative tokens during prediction.

2) For the LSTM Model: To ensure the LSTM model is trained on a rich and varied vocabulary, the initial dataset was constructed by combining two randomly chosen text files from the Project Gutenberg corpus: “Around the World in Eighty Days” and “The Evolution of Love.”

Text Cleaning: After loading and concatenating these sources, a comprehensive text cleaning process is applied to standardize the data. This involved several steps: removing all special characters and digits; eliminating newline characters, dashes, and extraneous spaces; and converting the entire text to lowercase.

Tokenization: A character-level vocabulary is established by identifying all unique characters present in the preprocessed text. Subsequently, a bijective mapping is created to assign a unique integer index to each character.

Input-Output Generation: This step is accomplished using a sliding window approach, where a fixed-length sequence of 100 characters was extracted to serve as the input feature X, and the single character immediately following this sequence was designated as the target label y. This window was moved sequentially across the entire dataset, generating a comprehensive set of training examples where the model learns to predict the next character based on the preceding context.

Vectorization: The data is formatted to meet the specific input requirements of the Keras LSTM architecture. The input sequences is reshaped into a 3D tensor with dimensions corresponding to samples, timesteps, and features. To ensure stable and efficient training, the input features were normalized to a range between 0 and 1. The categorical target labels were converted into a one-hot encoded format for multi-class classification.

IV. Algorithms

A. N-Gram Language Models

The N-gram model is a simple and interpretable statistical language model that predicts the next word in a sequence by looking at the previous n-1 words. This implementation supports unigram, bigram, and trigram models using a general-purpose class structure. The model builds a dictionary of observed n-grams from a training corpus and uses frequency counts to make predictions.

Given a sequence of words $w_1, w_2, \ldots, w_T$, the N-gram approximation of the joint probability is:

$$P(w_1, w_2, \ldots, w_T) \approx \prod_{t=1}^{T} P(w_t \mid w_{t-n+1}^{t-1})$$

as described in [8].

For example, in a trigram model ($n = 3$):

$$P(w_t \mid w_{t-2}, w_{t-1})$$

Preprocessing: The texts convert to lowercase, tokenize using NLTK's tokenizer, and cleans by removing punctuation and non-alphabetic tokens.

Padding: Each sentence prepends $(n-1)$ `<START>` tokens and appends a `<END>` token to preserve boundary context.

N-Gram Extraction: For each tokenized sequence, the model extracts all possible n-grams. Each n-gram splits into a prefix and a next word:

$$\text{prefix} = (w_{t-n+1}, \ldots, w_{t-1}), \quad \text{next word} = w_t$$

The model counts the frequency of each $(\text{prefix}, \text{next word})$ pair using a nested dictionary.

Prediction and Backoff Strategy: Given an input prompt, the model extracts the last $(n-1)$ tokens as context and looks up the most frequent next words:

$$\text{Prediction: } \hat{w} = \arg\max_{w} P(w \mid w_{t-n+1}^{t-1})$$

If the prefix is unseen, the model backs off to shorter prefixes (e.g., bigram, unigram) to estimate probabilities. If no match is found even in lower-order models, a random word from the vocabulary is selected.

Sentence Generation: To generate a sentence, the model begins with a prefix of $(n-1)$ `<START>` tokens. At each step, it samples the next word using the learned distribution, as defined in [9]:

$$P(w \mid \text{prefix}) = \frac{\text{count}(\text{prefix}, w)}{\sum_{w'} \text{count}(\text{prefix}, w')}$$

The model continues this process until generating an `<END>` token or reaching a maximum word limit. It then formats the generated sentence with capitalization and punctuation.

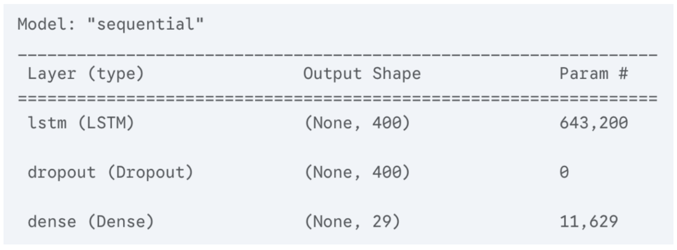
Evaluation Metrics: We evaluate our N-gram models using perplexity, which measures how well a language model predicts a sequence of words. Lower perplexity indicates better performance. Given a test sequence $w_1, w_2, \ldots, w_T$, perplexity is defined as [10]:

$$\text{Perplexity} = \exp\left(-\frac{1}{T}\sum_{t=1}^{T} \log P(w_t \mid w_{t-n+1}^{t-1})\right)$$

Here, T represents the number of words, and $P(w_t \mid w_{t-n+1}^{t-1})$ is the conditional probability the model estimates. This metric compares performance across different values of n.

B. Long Short-Term Memory Models

1) Architectures of Models: To investigate the architectural impacts on model performance, we design and implement three distinct LSTM network architectures. These models are systematically varied to explore the effects of scaling out (increasing the width of a layer) versus scaling up (increasing the depth of the network). All models share a common structure: they take sequences of 100 characters as input, are regularized with a $dropout$ rate of 0.2 to prevent overfitting, and use a $dense$ (fully connected) layer with a $softmax$ activation function to predict the next character from a vocabulary of 29 unique characters.

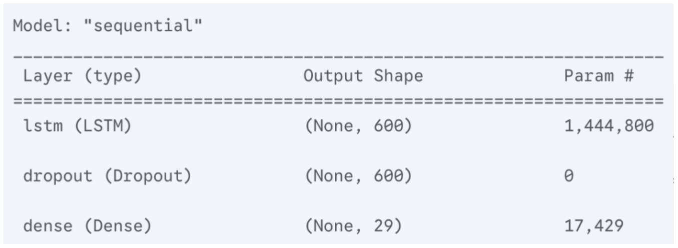

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 400)               643,200

 dropout (Dropout)           (None, 400)               0

 dense (Dense)               (None, 29)                11,629
```

Fig. 3. Baseline LSTM Model Structure

Baseline Model: The baseline model serves as the foundational architecture for our experiments. It consists of a single LSTM layer with 400 hidden units. This standard configuration establishes a performance benchmark against which the scaled models can be compared. Its architecture is shown in Figure 3.

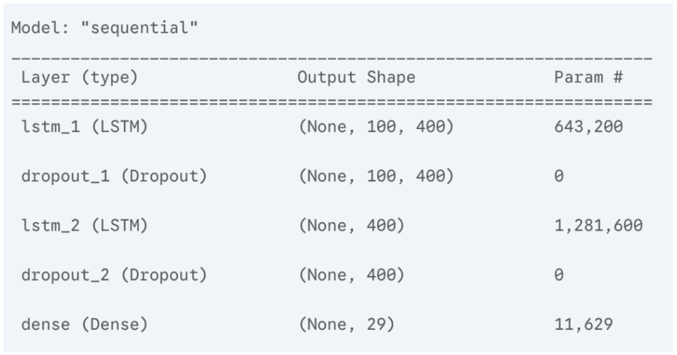

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 600)               1,444,800

 dropout (Dropout)           (None, 600)               0

 dense (Dense)               (None, 29)                17,429
```

Fig. 4. Wider LSTM Model Structure

Wider Model (Scaling Out): To explore the effect of scaling out, the wider model increases the representational capacity of the network by expanding the width of the single LSTM layer. In Figure 4, the number of hidden units in the LSTM layer increases by 50%, from 400 to 600. This modification allows the model to learn more complex patterns from the data within a single layer but does not change the network's depth.



```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_1 (LSTM)               (None, 100, 400)          643,200

 dropout_1 (Dropout)         (None, 100, 400)          0

 lstm_2 (LSTM)               (None, 400)               1,281,600

 dropout_2 (Dropout)         (None, 400)               0

 dense (Dense)               (None, 29)                11,629
```

Fig. 5. Deeper LSTM Model Structure

Deeper Model (Scaling Up): To investigate the effect of scaling up, the deeper model increases the network's depth by stacking two LSTM layers. This approach allows for hierarchical feature extraction, where the first layer learns basic patterns from the sequence and the second layer learns more abstract patterns from the output of the first. Each LSTM layer contains 400 units, maintaining the same width as the baseline model but doubling the network's depth as appeared in Figure 5.

2) Optimization Algorithm: The models are trained by minimizing the categorical cross-entropy loss function. As the standard loss function for multi-class classification problems, it measures the dissimilarity between the predicted probability distribution and the true distribution (the one-hot encoded actual character), providing a robust gradient for the optimization process.

$$L(y, \hat{y}) = -\sum_{i=1}^{C} y_i \log(\hat{y}_i) \tag{1}$$

To minimize this loss, we employ the Adam (Adaptive Moment Estimation) optimization algorithm. The model's weights are updated using a mini-batch gradient descent approach with a batch size of 50, meaning the model processes 50 training examples at a time before updating its parameters. This provides a balance between computational efficiency and rapid convergence, and the Adam optimizer complements this process by adapting the learning rate for each model parameter individually.

3) Evaluation Metrics: The primary metric used to evaluate and compare model performance during training and validation is accuracy. In the context of this character-level language model, accuracy represents the percentage of times the model correctly predicts the next character in a sequence. This provides a direct and interpretable measure of the model's predictive capability.

V. Result and Analysis

A. N-Gram Language Models

Efficiency and Limitations: This implementation uses a dictionary structure to efficiently store and retrieve n-gram frequencies. However, the model does not implement smoothing techniques such as Laplace or Kneser-Ney smoothing, which may result in zero probabilities for unseen sequences. Despite this, the model is computationally lightweight and interpretable, making it suitable for sentence completion tasks on small- to medium-sized corpora.

Training Performance: We evaluate unigram, bigram, and trigram models using perplexity as a measure of predictive performance on a held-out test set. Contrary to typical expectations, perplexity increases with model order. This trend suggests that higher-order N-gram models suffer from data sparsity, especially given the domain-specific nature of the training corpus, which contains Shakespearean and Old English texts.

Table I reports the perplexity scores for each N-gram configuration. The unigram model achieves the lowest perplexity, while the trigram model performs the worst. This outcome indicates that the higher-order models frequently encounter unseen word combinations, making it difficult to assign reliable probabilities.

TABLE I
Perplexity Scores of N-Gram Models

Model	**Perplexity**
Unigram	**619.18**
Bigram	6037.59
Trigram	31118.15

Figure 6 visualizes the perplexity scores. The sharp increase highlights the limitations of relying on longer contexts when the training data lacks sufficient coverage of phrase patterns. The N-gram models struggle to generalize across the syntactic and lexical diversity present in early modern English.

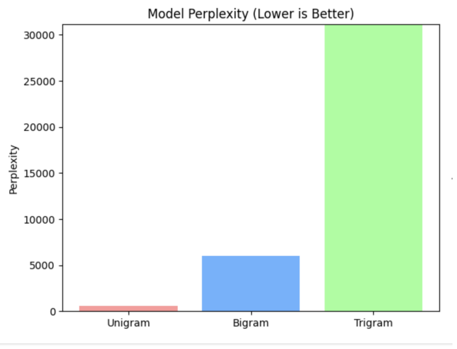


Fig. 6. Perplexity Scores for Unigram, Bigram, and Trigram Models

These results emphasize a core limitation of traditional N-gram models: their reliance on memorized sequences. In domains with archaic vocabulary or low overlap between

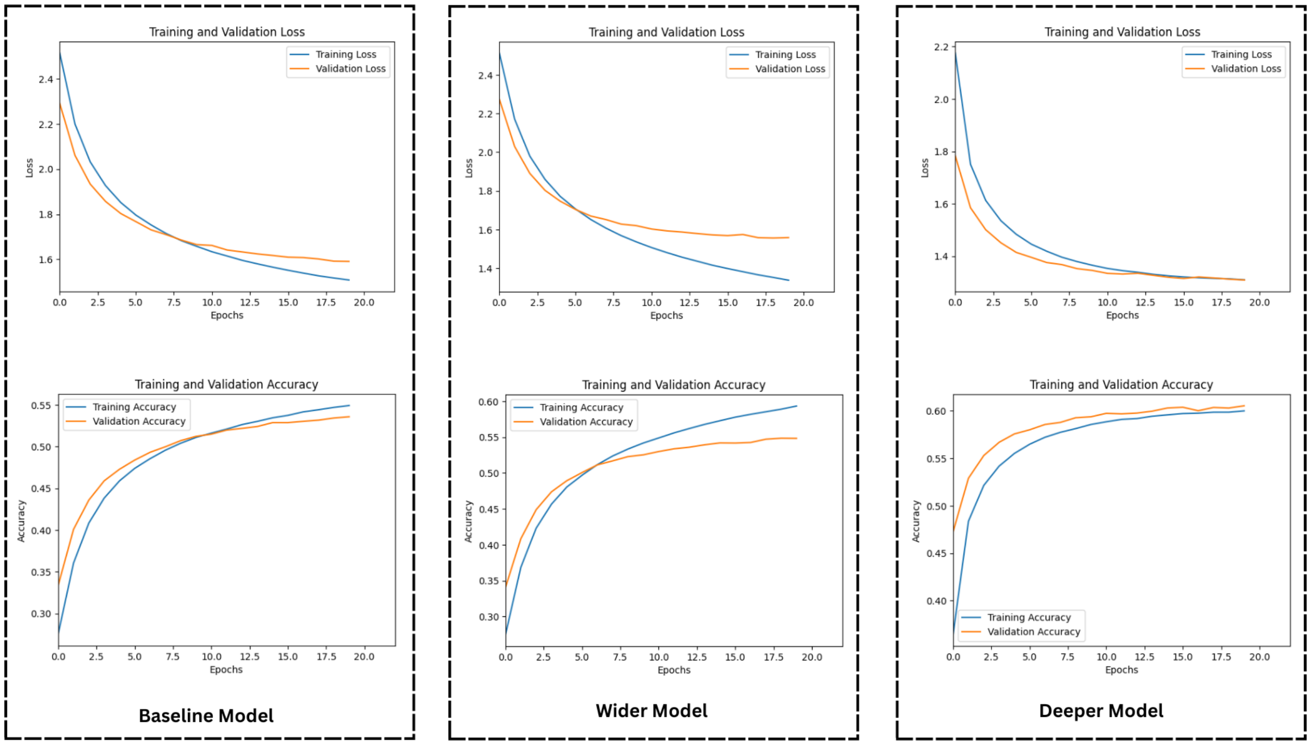


Fig. 7. Training and Validation Performance of LSTM Models

training and test distributions, higher-order models tend to overfit sparse contexts, leading to poor generalization [11].

B. Long Short-Term Memory Models

1) Training Performance: The dataset is first partitioned into an 80% training set and a 20% validation set. All three models are then trained for 20 epochs to ensure a consistent basis for comparison. Figure 7 reveals significant insights into how architectural choices affect learning and generalization

The *Baseline Model* shows steady improvement, with the training and validation loss curves decreasing in tandem. The accuracy curves for both training and validation also rise closely together, indicating that the model is learning to generalize from the training data without significant overfitting. It achieves a final validation accuracy of 53.57%.

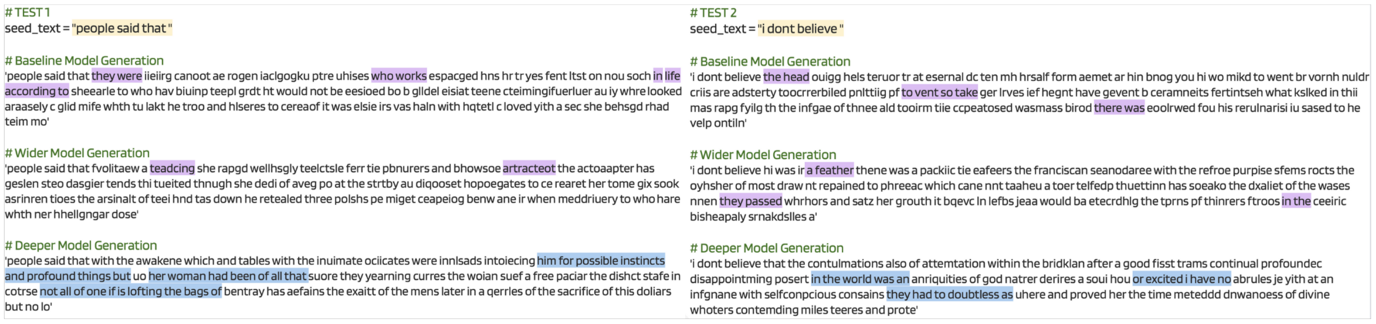
The *Wider Model*, with its increased capacity in a single layer, learns faster on the training data, achieving a higher training accuracy (59.84%) than the baseline. However, a noticeable gap emerges between the training and validation curves for both loss and accuracy. The validation loss plateaus and even begins to increase slightly towards the final epochs, while the validation accuracy flattens out at 54.83%. This divergence is a classic sign of overfitting, where the model begins to memorize the training data rather than learning the underlying patterns, thus failing to generalize effectively to the unseen validation data.

The *Deeper Model* demonstrates the most effective learning pattern. It learns much faster than the baseline in the initial epochs and ultimately achieves the highest validation accuracy (60.54%) and the lowest validation loss. Crucially, the training and validation curves remain very close throughout the 20 epochs, and in some cases, the validation performance is slightly better than the training performance. This indicates excellent generalization. The hierarchical structure allows the model to learn more complex and robust features, and the use of Dropout at each layer effectively regularizes the model, preventing the overfitting seen in the wider architecture. This confirms that for this task, scaling up (depth) provides a more significant and stable performance improvement than scaling out (width).

2) Text Generation: To qualitatively assess the performance of the three architectures, two generation tests are conducted using the seed texts "*people said that*" and "*i dont believe*". The results in Figure 8 highlight the significant impact of model depth on the coherence and quality of the generated text.

The text generated by the baseline model is largely incoherent and consists of misspelled or non-existent words. For both seed texts, it produces output that fails to form a cohesive sentence structure. This indicates that while the model has learned some basic character-level associations, it lacks the capacity to form coherent words or grammatical phrases.

The wider model shows a marginal improvement. While still mostly gibberish, it produces more recognizable, albeit misspelled, word-like tokens (e.g., "*teadcing*," "*artracteot*," "*feather*"). This suggests its increased capacity allows it to



```
# TEST 1
seed_text = "people said that "

# Baseline Model Generation
'people said that they were iieiirg canoot ae rogen iaclgogku ptre uhises who works espacged hns hr tr yes fent ltst on nou soch in life
according to sheearle to who hav biuinp teepl grdt ht would not be eesioed bo b glldel eisiat teene cteimingifuerluer au iy whre looked
araasely c glid mife whth tu lakt he troo and hlseres to cereaof it was elsie irs vas haln with hqtetl c loved yith a sec she behsgd rhad
teim mo'

# Wider Model Generation
'people said that fvolitaew a teadcing she rapgd wellhsgly teelctsle ferr tie pbnurers and bhowsoe artracteot the actoaapter has
geslen steo dasgier tends thi tueited thnugh she dedi of aveg po at the strtby au diqooset hopoegates to ce rearet her tome gix sook
asrinren tioes the arsinalt of teei hnd tas down he retealed three polshs pe miget ceapeiog benw ane ir when meddriuery to who hare
whth ner hhellgngar dose'

# Deeper Model Generation
'people said that with the awakene which and tables with the inuimate ociicates were innlsads intoiecing him for possible instincts
and profound things but uo her woman had been of all that suore they yearning curres the woian suef a free paciar the dishct stafe in
cotrse not all of one if is lofting the bags of bentray has aefains the exaitt of the mens later in a qerrles of the sacrifice of this doliars
but no lo'
```

```
# TEST 2
seed_text = "i dont believe "

# Baseline Model Generation
'i dont believe the head ouigg hels teruor tr at esernal dc ten mh hrsalf form aemet ar hin bnog you hi wo mikd to went br vornh nuldr
criis are adsterty toocrrerbiled pnlttiig pf to vent so take ger lrves ief hegnt have gevent b ceramneits fertintseh what kslked in thii
mas rapg fyilg th the infgae of thnee ald tooirm tiie ccpeatosed wasmass birod there was eoolrwed fou his rerulnarisi iu sased to he
velp ontiln'

# Wider Model Generation
'i dont believe hi was ir a feather thene was a packiic tie eafeers the franciscan seanodaree with the refroe purpise sfems rocts the
oyhsher of most draw nt repained to phreeac which cane nnt taaheu a toer telfedp thuettinn has soeako the dxaliet of the wases
nnen they passed whrhors and satz her grouth it bqevc ln lefbs jeaa would ba etecrdhlg the tprns pf thinrers ftroos in the ceeiric
bisheapaly srnakdslles a'

# Deeper Model Generation
'i dont believe that the contulmations also of attemtation within the bridklan after a good fisst trams continual profoundec
disappointming posert in the world was an anriquities of god natrer derires a soui hou or excited i have no abrules je yith at an
infgnane with selfconpcious consains they had to doubtless as uhere and proved her the time meteddd dnwanoess of divine
whoters contemding miles teeres and prote'
```

Fig. 8. Text Generation of LSTM Models

learn slightly more complex character combinations. However, it fails to connect these tokens into meaningful phrases, demonstrating that simply increasing the width of a single layer does not significantly improve the model's grasp of language structure.

The deeper model's output is qualitatively superior by a large margin. It generates grammatically plausible phrases and demonstrates an understanding of context and sentence structure. For example, it produces coherent fragments like "*for possible instincts and profound things*" and "*they had to doubtless as uhere and proved her*." This model clearly benefits from its hierarchical structure, where the first LSTM layer learns word fragments and the second layer learns to assemble them into more complex and meaningful sequences. This result strongly supports the conclusion that depth is a more critical factor than width for learning the complex dependencies of natural language.

It is important to note that a character-based model inherently faces a more complex task than a word-based model. It must first learn to structure characters into coherent words, and only then learn to arrange those words into grammatically correct sentences. A word-based model, by contrast, can focus directly on sentence structure. The trade-off for this simpler task is a significantly larger output vocabulary (tens of thousands of words versus 29 characters), which can make a word-based model much larger and slower to train.

VI. Conclusion

This project compares statistical N-gram models and neural LSTM networks for the task of sentence auto-completion. Both models aim to predict the most likely continuation of a sentence, but differ significantly in design and performance.

The N-gram models, while fast and interpretable, show clear limitations when handling sparse or unseen data. As model order increases, so does perplexity, highlighting the model's struggle with long-range dependencies and rare word sequences. They perform reasonably well on frequent patterns and structured language, but are held back by their limited context window, which is typically 2 to 4 tokens.

LSTM models offer better results, designed to learn from much longer sequences. Deeper architectures, which enable hierarchical learning, significantly outperform both shallow and wide configurations. The LSTM model not only achieves the highest validation accuracy, but also possesses better resilience to overfitting. However, LSTM models also require more computational resources and training data, making them less practical for small-scale applications.

Ultimately, this project highlights the importance of model selection based on task complexity and available resources. Although N-grams offer simplicity and speed, LSTMs provide a more powerful framework for capturing the nuanced patterns present in natural language.

Future work could extend this comparison to include state-of-the-art transformer-based models. For the N-gram model, incorporating smoothing techniques could help address data sparsity issues. Collectively, these enhancements could enable even richer sentence generation performance. For the LSTM models, several avenues for improvement exist. One could explore more advanced methods of deepening the architecture, such as incorporating a bidirectional LSTM to enable the model to consider both past and future context. Adopting a word-based tokenization approach, while computationally more expensive, could also allow the model to learn sentence structure more directly.

References

[1] Gadesha and E. Kavlakoglu, "What is text generation?," IBM Think, Mar. 19, 2024. [Online]. Available: https://www.ibm.com/think/topics/text-generation. [Accessed: Jul. 29, 2025].

[2] D. Jurafsky and J. H. Martin, Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition. Prentice Hall, 2000.

[3] S. Hochreiter and J. Schmidhuber, "Long short-term memory," Neural computation, vol. 9, no. 8, pp. 1735-1780, 1997.

[4] A. Vaswani et al., "Attention is all you need," in Advances in Neural Information Processing Systems, 2017, pp. 5998–6008.

[5] Project Gutenberg, n.d. [Online.] Available: https://www.gutenberg.org/

[6] N. Saeed. "Sentence Auto-Completion Using LSTM Deep Learning." Medium, Apr. 14, 2024 [Online]. Available: https://medium.com/@611noorsaeed/sentence-auto-completion-using-lstm-deep-learning-4b6973d7786a. [Accessed Jun 29, 2025].

[7] P. Srivastava, "Text Generators," 2018, Github Repository. [Online]. Available: https://github.com/pranjal52/text-generators/tree/master. [Accessed Jun 29, 2025].

[8] O. Borisov, "Text generation using N-Gram model," *Towards Data Science*, Oct. 27, 2020. [Online]. Available: https://towardsdatascience.com/text-generation-using-n-gram-model-8d12d9802aa0/. [Accessed: Jul. 30, 2025].

[9] A. Jain, "N-grams in NLP," *Medium*, Feb. 5, 2024. [Online]. Available: https://medium.com/@abhishekjainindore24/n-grams-in-nlp-a7c05c1aff12. [Accessed: Jul. 30, 2025].

[10] V. S. Ağzıyağlı, “Text generation using N-gram,” *Medium*, May 19, 2022. [Online]. Available: https://medium.com/@vsagziyagli/text-generation-using-n-grams-ef49e6e43d39. [Accessed: Jul. 30, 2025].

[11] S. Sadar, “Text Generation Using N-Gram Models,” 2019, Github Repository. [Online]. Available: https://github.com/shayan09/ngam-text-generation. [Accessed Jun 29, 2025].

[12] P. Srivastava, “Text Generators,” 2018, Github Repository. [Online]. Available: https://github.com/pranjal52/text-generators/tree/master. [Accessed Jun 29, 2025].

[13] D. Jurafsky and J. H. Martin, ”N-Gram Language Models,” in *Speech and Language Processing*. Harlow: Pearson Education, 2014.

[14] E. M. de Novais, T. D. Tadeu, and I. Paraboni, “Improved Text Generation Using N-gram Statistics,” in *Iberamia*, Springer-Verlag Berlin Heidelberg, 2010. Accessed: Jul. 30, 2025. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-642-16952-6_32

[15] F. Chiusano, “Text Generation with N-Grams,” NLPlanet, 2022. Available:https://www.nlplanet.org/course-practical-nlp/01-intro-to-nlp/08-text-generation-n-grams. [Accessed Jul. 30, 2025].