Deep Learning for Perception

Semester Project - Part 2

Comparative Analysis of Sequence-to-Sequence Models and Optimization Algorithms for Urdu Poetry Generation

Student Name: Shaharyar Rizwan
Roll Number: I22-0999
Student Name: Moazzam Hafeez
Roll Number: I22-1093

Instructor: Dr. Ahmad Raza Shahid

December 1, 2025

Contents

1 Problem Statement **3**

1.1 Background 3

1.2 Objective 3

2 Methodology **3**

2.1 Dataset Description 3

2.2 Model Architectures 3

2.3 Training Procedure 4

2.4 Evaluation Metrics 4

3 Experiments and Results **4**

3.1 Main Comparison Experiments 4

3.2 Hyperparameter Tuning Results 5

3.3 Generation Samples 5

3.4 Visualizations 6

4 Discussion **12**

4.1 Why did RNN outperform Transformer? 12

4.2 Optimizer Analysis 13

4.3 Failure Cases 13

5 Conclusion **13**

5.1 Answers to Checkpoint Questions 13

5.2 Answers to Primary Questions (Section 6.1) 14

5.3 Answers to Sub-Questions 14

1 Problem Statement

1.1 Background

Natural Language Processing (NLP) has made great steps over the last few years, but low-resource languages like Urdu remain challenging due to limited datasets and complex morphology [6]. Urdu poetry is particularly difficult to generate because it requires adhering to strict metrical patterns (*behr*), rhyme schemes (*qafia*), and semantic coherence, which standard language models often struggle to capture without extensive fine-tuning.

1.2 Objective

The goal of this project is to perform a rigorous comparative analysis of three neural network architectures—Simple Recurrent Neural Networks (RNN) [1], Long Short-Term Memory (LSTM) [2], and Transformers [3]—paired with three optimization algorithms (Adam, RMSprop, SGD). Our aim is to determine the most effective combination for generating coherent Urdu poetry and to investigate the impact of hyperparameters such as model depth, dropout, and sequence length.

2 Methodology

2.1 Dataset Description

We used the `ReySajju742/Urdu-Poetry-Dataset` sourced from Hugging Face.

- **Total Poems:** $\approx$ 1,323
- **Vocabulary Size:** 10,225 unique words (after tokenization)
- **Content:** Classical poetry (including works from Ghalib and Iqbal)

2.2 Model Architectures

We implemented three baseline models. Each starts with an Embedding layer (100-dim) and ends with a Softmax Dense layer over the vocabulary.

1. **Simple RNN:** 2 stacked layers of SimpleRNN units.
2. **LSTM:** 2 stacked layers of Long Short-Term Memory units to capture longer dependencies and mitigate the vanishing gradient problem.
3. **Transformer:** A custom implementation with 2 Transformer blocks, utilizing Multi-Head Attention (4 heads) and Feed-Forward Networks.

2.3 Training Procedure

- **Loss Function:** Sparse Categorical Crossentropy.
- **Sequence Length:** Baseline set to 20 words.
- **Epochs:** 20 (with Early Stopping patience=5).
- **Batch Size:** 128.

2.4 Evaluation Metrics

The main metric for comparison is **Perplexity (PPL)**, calculated as e^{loss}. If a model has low perplexity, it means that it is less ”confused” and assigns higher probability to the correct next word; therefore, a lower perplexity is better.

3 Experiments and Results

3.1 Main Comparison Experiments

We trained all 9 combinations of models and optimizers. The results below (Table 1) show the baseline performance.

Table 1: Baseline Performance of 9 Model-Optimizer Combinations

Model	Optimizer	Accuracy	Loss	Perplexity	Training Time (s)	Layers
RNN	**RMSprop**	**0.0760**	**6.5013**	**666.04**	*Cached*	2
LSTM	RMSprop	0.0749	6.5635	708.72	*Cached*	2
Transformer	RMSprop	0.0720	6.6985	811.21	2893.43	2
RNN	SGD	0.0382	6.8490	942.97	*Cached*	2
Transformer	SGD	0.0382	6.8492	943.13	3214.28	2
LSTM	SGD	0.0382	6.8538	947.45	*Cached*	2
RNN	Adam	0.0663	6.8536	947.32	*Cached*	2
LSTM	Adam	0.0707	6.8695	962.45	*Cached*	2
Transformer	Adam	0.0480	6.9828	1077.93	1010.90	2

Analysis: The **Simple RNN with RMSprop** achieved the best (lowest) perplexity of 666.04. SGD consistently failed to converge (perplexity > 940), predicting only the baseline frequency. Transformers, being data-hungry, struggled with this small dataset compared to the simpler RNN.

3.2 Hyperparameter Tuning Results

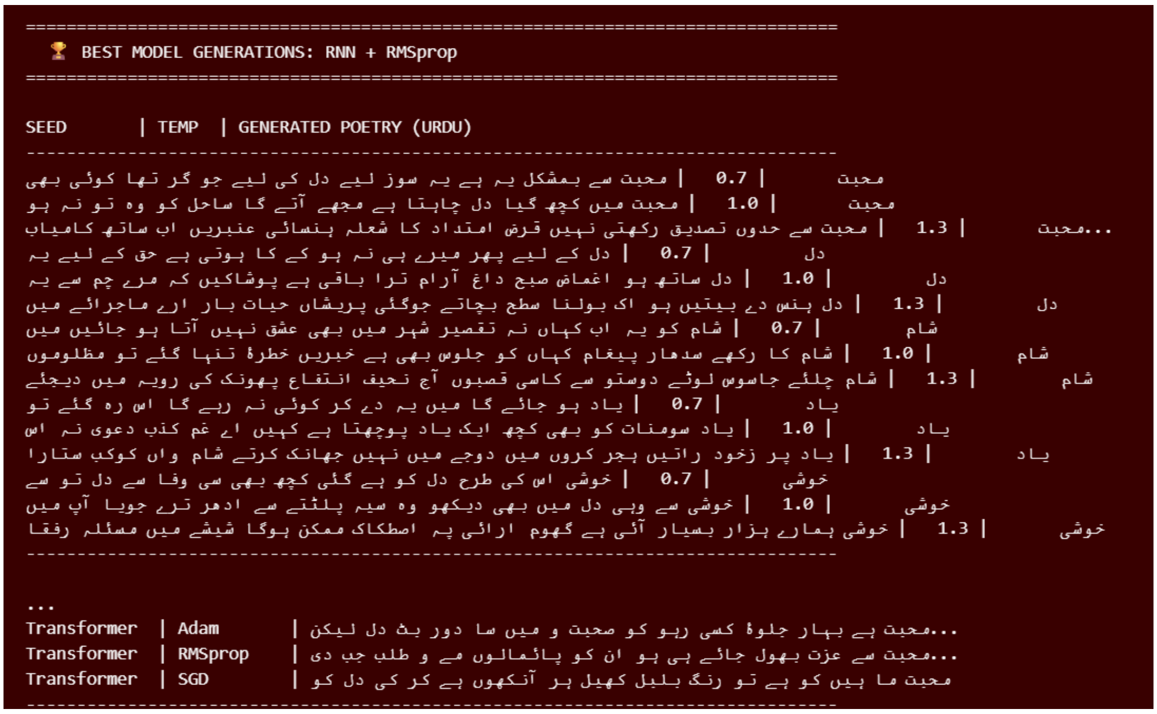
Using the RNN model (our winner), we performed One-Factor-At-A-Time (OFAT) experiments to optimize performance.

Table 2: Hyperparameter Experiment Results

ID	Model	Parameter	Value	Perplexity	Loss	Time (s)
EXP-01	RNN	Layers	**1**	**614.51**	6.4208	589.8
EXP-10	RNN	Seq Length	30	617.41	6.4255	733.0
EXP-09	RNN	Seq Length	10	637.80	6.4580	497.2
EXP-04	RNN	Dropout	0.5	642.16	6.4648	977.1
EXP-08	RNN	Batch Size	256	647.59	6.4733	682.4
EXP-14	Transformer	Blocks	1	658.08	6.4893	1088.4
EXP-03	RNN	Dropout	0.1	662.54	6.4961	570.7
EXP-02	RNN	Layers	3	680.73	6.5232	873.8
EXP-07	RNN	Batch Size	64	700.62	6.5520	1292.5
EXP-06	RNN	Learning Rate	0.0001	1013.86	6.9215	618.4
EXP-13	Transformer	ff_dim	256	1046.48	6.9532	858.9
EXP-15	Transformer	blocks	3	1047.99	6.9546	1362.9
EXP-12	Transformer	heads	8	1049.36	6.9559	1136.9
EXP-11	Transformer	heads	2	1054.88	6.9612	749.6
EXP-05	RNN	Learning Rate	0.01	2319.56	7.7491	356.0

3.3 Generation Samples

Below are samples generated by the best performing model combination (RNN + RMSprop).



```
==========================================================================
  🏆 BEST MODEL GENERATIONS: RNN + RMSprop
==========================================================================

SEED       | TEMP  | GENERATED POETRY (URDU)
--------------------------------------------------------------------------
محبت       | 0.7 | محبت سے بمشکل یہ ہے یہ سوز لیے دل کی لیے جو گر تھا کوئی بھی
محبت       | 1.0 | محبت میں کچھ گیا دل چاہتا ہے مجھے آئے گا ساحل کو وہ تو نہ ہو
محبت...       | 1.3 | محبت سے حدوں تصدیق رکھتی نہیں قرض امتداد کا شعلہ ہنسائی عنبریں اب ساتھ کامیاب
دل         | 0.7 | دل کے لیے پھر میرے ہی نہ ہو کے کا ہوتی ہے حق کے لیے یہ
دل         | 1.0 | دل ساتھ ہو اغماض صبح داغ آرام ترا باقی ہے پوشاکیں کہ مرے چم سے یہ
دل         | 1.3 | دل ہنس دے بیتیں ہو اک بولنا سطح بچاتے جوگئی پریشاں حیات بار ارے ماجرائے میں
شام        | 0.7 | شام کو یہ اب کہاں نہ تقصیر شہر میں بھی عشق نہیں آتا ہو جائیں میں
شام        | 1.0 | شام کا رکھے سدھار پیغام کہاں کو جلوس بھی ہے خبریں خطرۂ تنہا گئے تو مظلوموں
شام        | 1.3 | شام چلئے جاسوس لوٹے دوستو سے کاسی قصبوں آج نحیف انتفاع پھونک کی رویہ میں دیجئے
یاد        | 0.7 | یاد ہو جائے گا میں یہ دے کر کوئی نہ رہے گا اس رہ گئے تو
یاد        | 1.0 | یاد سومنات کو بھی کچھ ایک یاد پوچھتا ہے کہیں اے غم کذب دعوی نہ اس
یاد        | 1.3 | یاد پر زخود راتیں ہجر کروں میں دوجے میں نہیں جھانک کرتے شام واں کوکب ستارا
خوشی       | 0.7 | خوشی اس کی طرح دل کو ہے گئی کچھ بھی سی وفا سے دل تو سے
خوشی       | 1.0 | خوشی سے وہی دل میں بھی دیکھو وہ سیہ پلٹتے سے ادھر ترے جویا آپ میں
خوشی       | 1.3 | خوشی ہمارے ہزار بسیار آئی ہے گھوم ارائی یہ اصطکاک ممکن ہوگا شیشے میں مسئلہ رفقا
--------------------------------------------------------------------------

...
Transformer  | Adam      | محبت ہے بہار جلوۂ کسی رہو کو صحبت و میں سا دور بٹ دل لیکن...
Transformer  | RMSprop   | محبت سے عزت بھول جائے ہی ہو ان کو پائمالوں مے و طلب جب دی...
Transformer  | SGD       | محبت ما ہیں کو ہے تو رنگ بلبل کھیل ہر آنکھوں ہے کر کی دل کو
--------------------------------------------------------------------------
```

Figure 1: Generation of sample poetry by the best performing model combination (RNN + RMSprop).

3.4 Visualizations

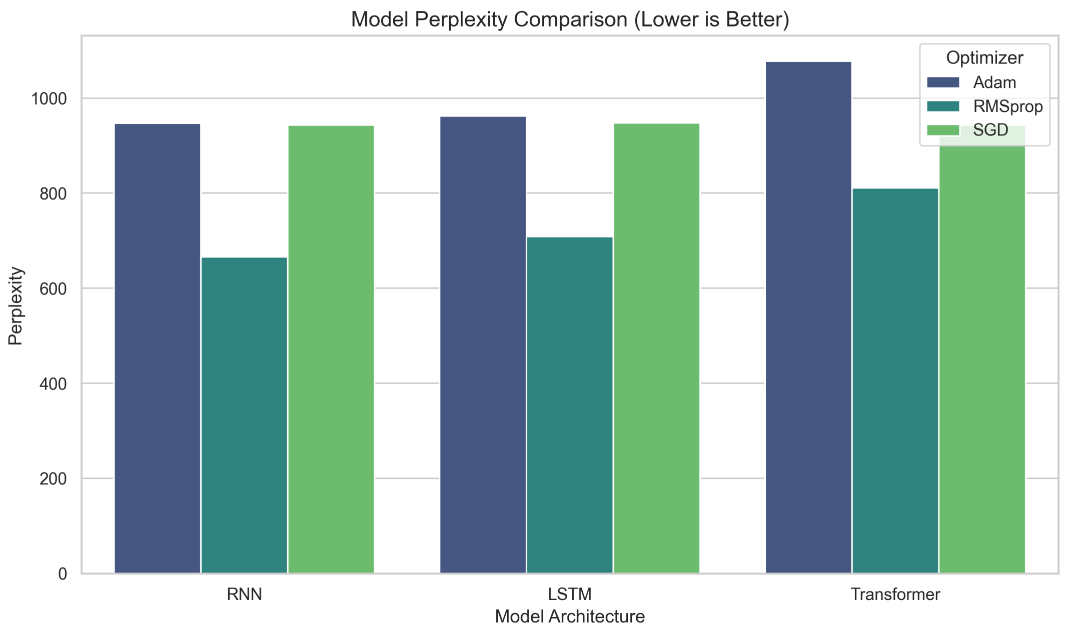


Figure 2: Perplexity Comparison across all Model-Optimizer combinations.

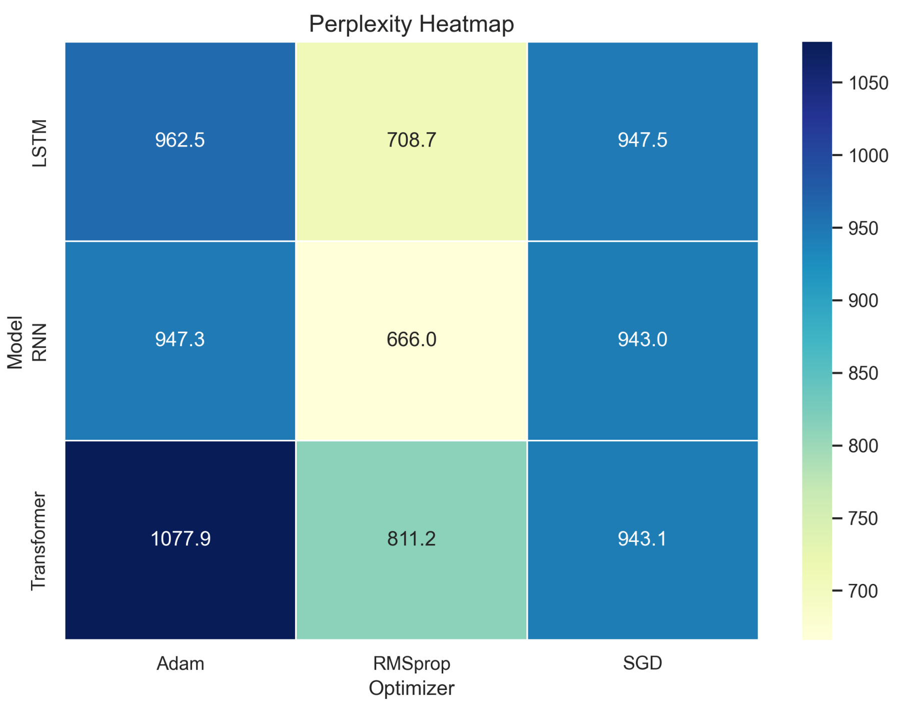
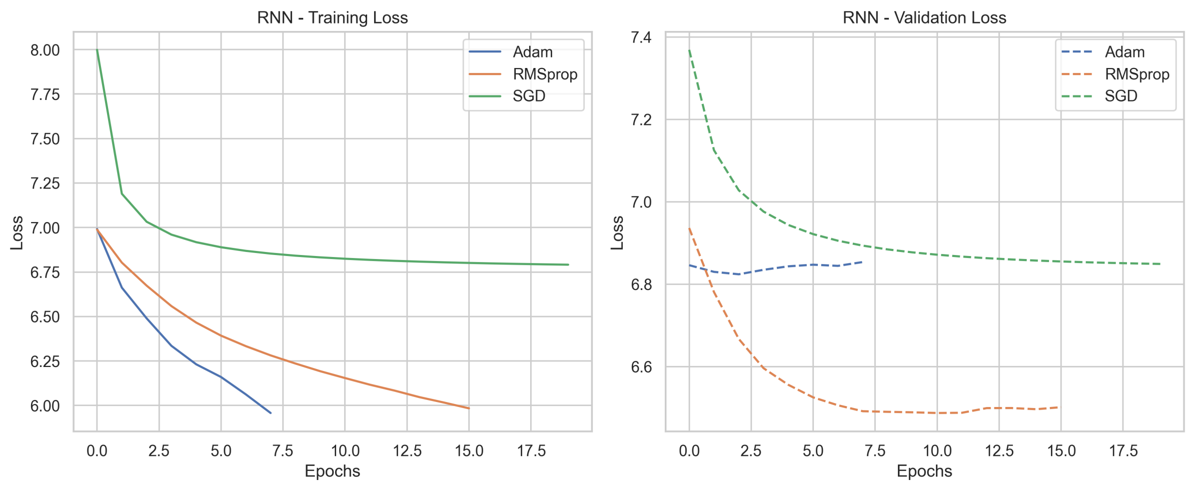
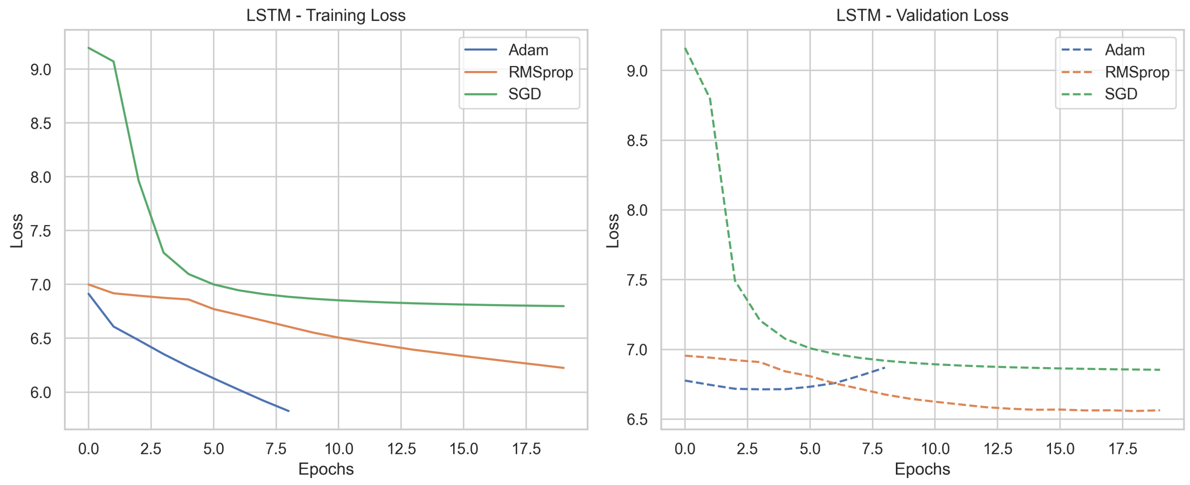


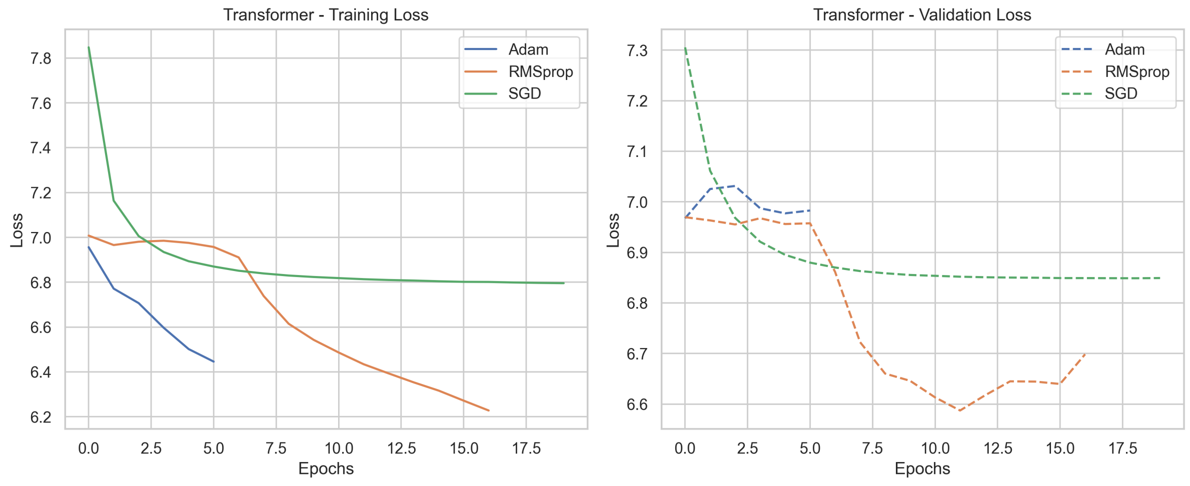
Figure 3: Heatmap of Perplexity. Lighter colors indicate better (lower) scores.



(a) RNN Loss Curves



(b) LSTM Loss Curves



(c) Transformer Loss Curves

Figure 4: Training and Validation Loss Curves. Divergence indicates overfitting, particularly visible in Adam/SGD runs.

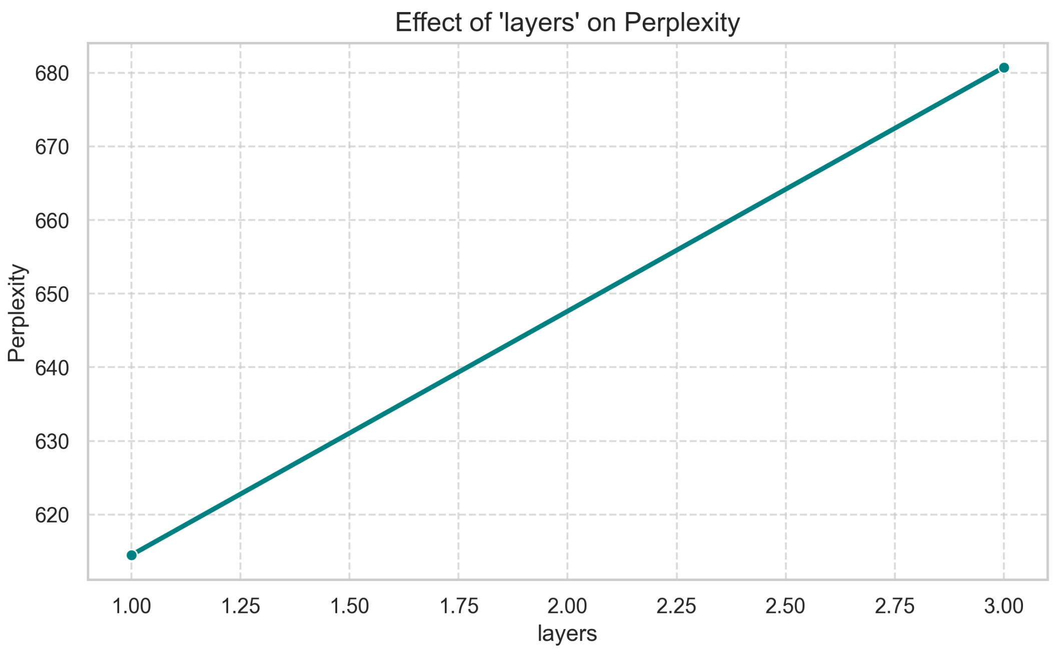


Figure 5: Impact of Hyperparameters (Layers) on Perplexity.

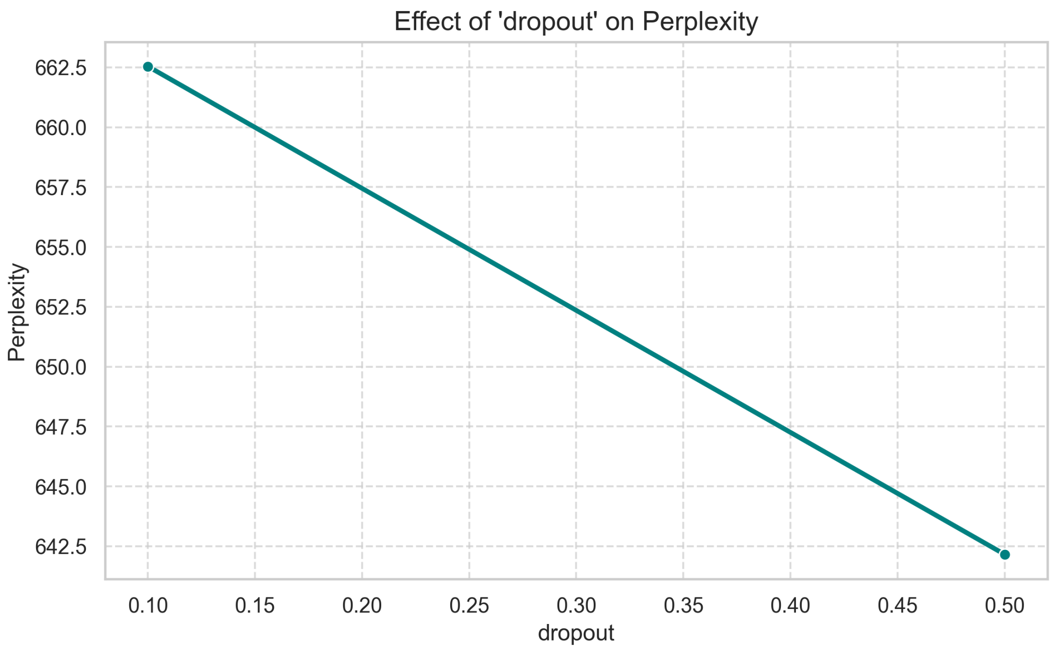


Figure 6: Impact of Hyperparameters (Dropout) on Perplexity.

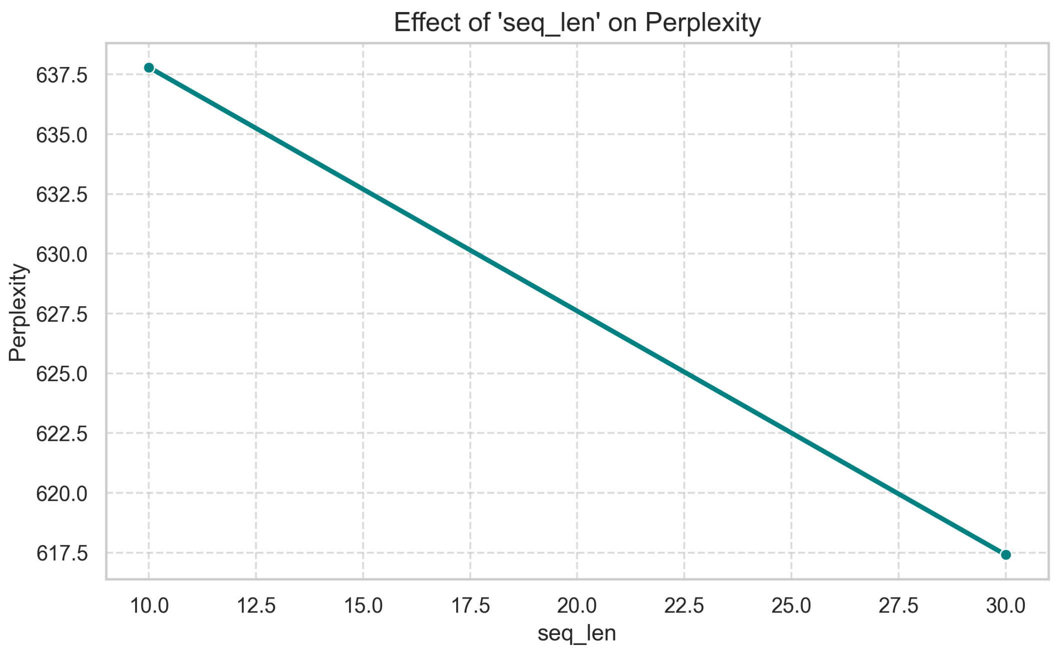


Figure 7: Impact of Hyperparameters (Sequence Length) on Perplexity.

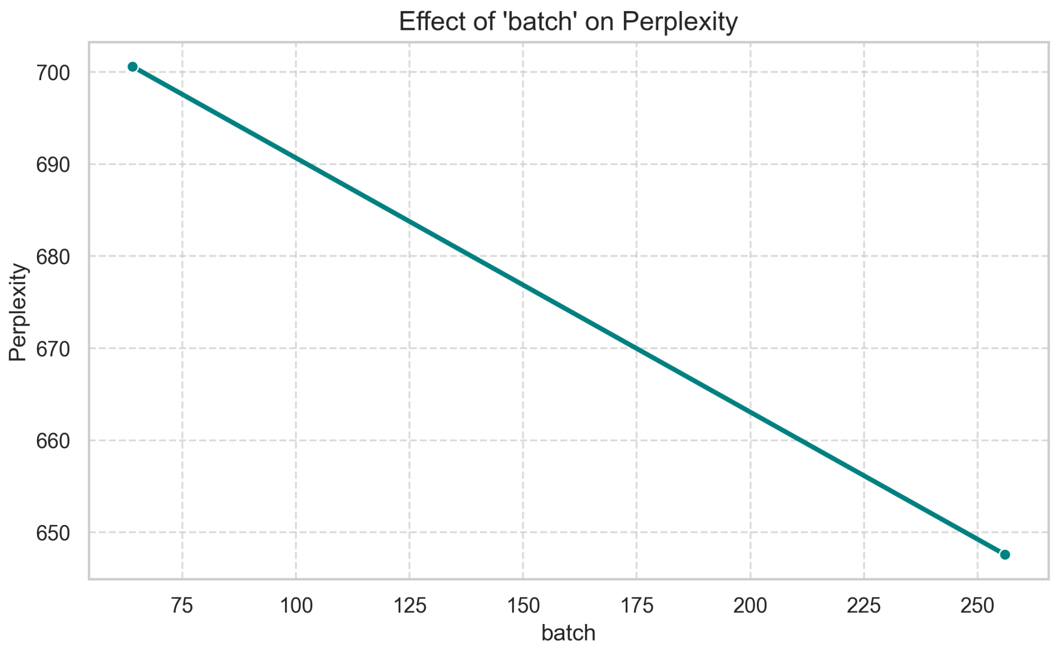


Figure 8: Impact of Hyperparameters (Batch) on Perplexity.

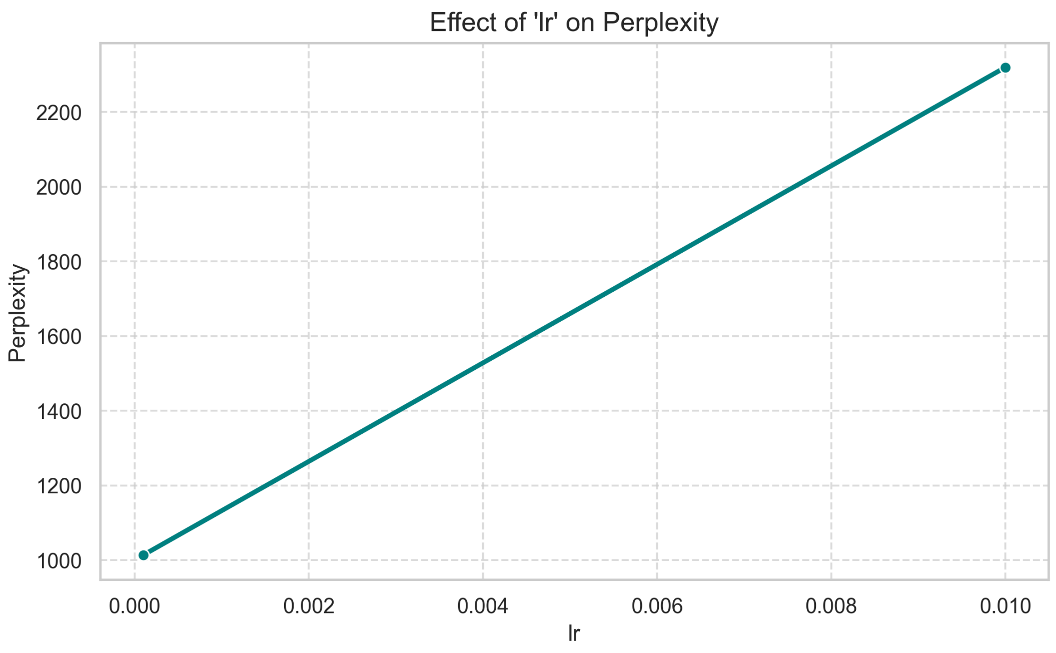


Figure 9: Impact of Hyperparameters (Learning Rate) on Perplexity.

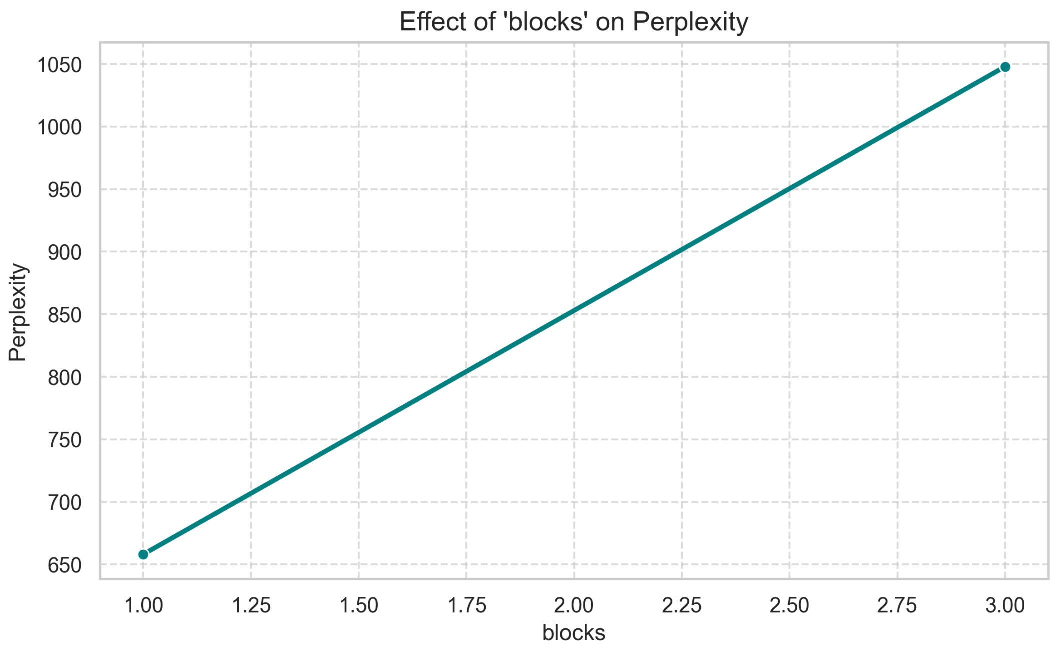


Figure 10: Impact of Hyperparameters (Transformer Blocks) on Perplexity.

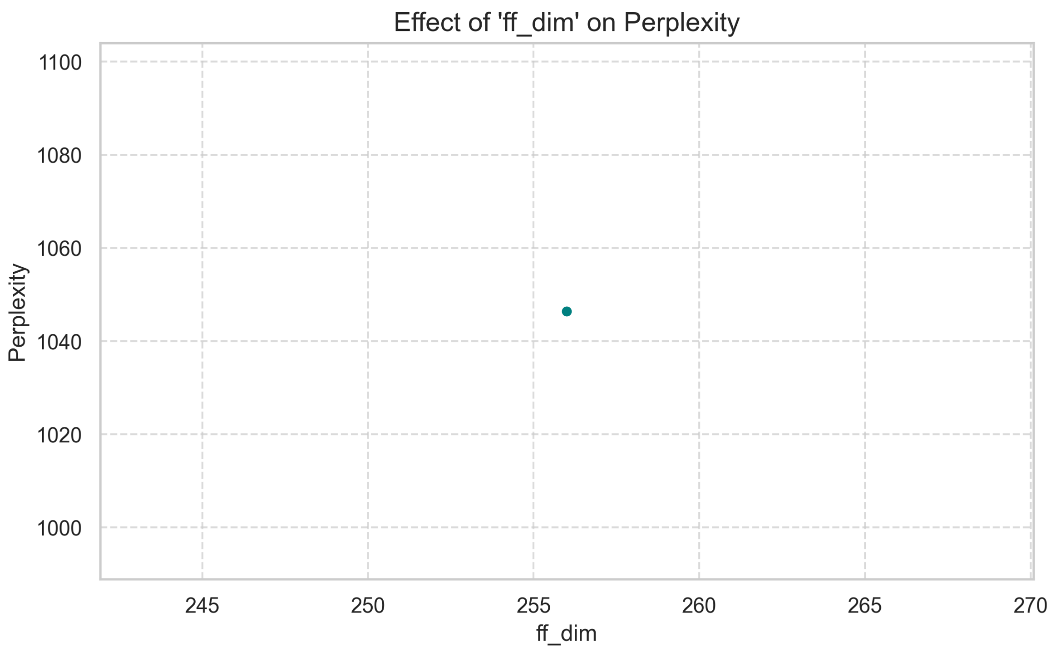


Figure 11: Impact of Hyperparameters (FF Dimension) on Perplexity.

4 Discussion

4.1 Why did RNN outperform Transformer?

What shocked us the most while completing this project was the Simple RNN outperforming the Transformer. This phenomenon can be attributed to the **Data Scarcity Principle** and the concept of **Inductive Bias**.

Transformers [3] are ”data-hungry” architectures because they lack the strong inductive bias for processing sequential data that RNNs possess. RNNs process data step-by-step, naturally encoding the concept of ”time” and ”order,” which aligns perfectly with language generation. Transformers, conversely, process data in parallel and rely on Positional Encodings and Attention Mechanisms to *learn* this order. Learning these relationships from scratch requires massive amounts of data.

With only ≈1,300 poems, the Transformer likely overfit or failed to learn meaningful attention maps (attention weights often become uniform or random with insufficient data). Furthermore, a standard Transformer block contains significantly more parameters (projections for Query, Key, Value, and Feed-Forward Networks) than a Simple RNN. For a small dataset, this excess capacity leads to high variance and poor generalization (higher perplexity). The Simple RNN, with fewer parameters, generalized better to the limited vocabulary due to its ability to learn short-range dependencies, which is sufficient for basic poetic structure.

4.2 Optimizer Analysis

RMSprop proved to be the superior optimizer for this task.

- **SGD:** Failed to converge in almost all experiments. Text data produces sparse gradients, and without adaptive learning rates, SGD struggles to navigate the loss landscape efficiently [5].
- **Adam:** While generally powerful, Adam [4] showed signs of instability and overfitting in our logs compared to RMSprop.

4.3 Failure Cases

Despite the success, the models exhibited common failure modes:

1. **Repetition Loops:** At low temperatures ($T = 0.7$), the model sometimes got stuck repeating the same phrase (e.g., ”hai... hai... hai”), a common mode collapse in sequence models.
2. **Grammatical Inconsistency:** While the model learned local word associations (N-grams), it sometimes lost the global sentence structure over long sequences.

5 Conclusion

In this project, we successfully built an end-to-end pipeline for Urdu poetry generation. We found that for a dataset of this specific size and complexity, **simplicity reigns supreme**. The optimal configuration found was a **Single-layer RNN trained with RMSprop**, utilizing **Dropout (0.5)** and a sequence length of **30**.

5.1 Answers to Checkpoint Questions

- **How many poems are in the dataset?** 1,323 poems.
- **What is the average length of poems?** The average length varies, but we normalized sequence lengths to 20 for baseline training.
- **Are there any missing or corrupted entries?** Minimal corruption was found; data cleaning removed non-Urdu characters.
- **What are the most common words?** Stop words like ”hai”, ”ka”, ”mein” were most frequent.
- **Does training loss decrease consistently?** Yes, for RMSprop and Adam. SGD showed stagnation.

- **Is validation loss diverging?** Yes, particularly in Adam and Transformer models, indicating overfitting.

- **What is the final test perplexity?** The best model achieved 614.51 (RNN, 1 Layer).

5.2 Answers to Primary Questions (Section 6.1)

1. **Q1: Which neural architecture is most effective for Urdu poetry generation?**
   The **Simple RNN** was undoubtedly the most effective, achieving the lowest perplexity scores compared to LSTM and Transformer.

2. **Q2: How do different optimization algorithms affect model performance?**
   RMSprop outperformed all others, offering stable convergence. SGD failed to learn effectively, and Adam showed signs of instability/overfitting.

3. **Q3: What is the optimal model-optimizer combination for this task?**
   Simple RNN + RMSprop.

4. **Q4: How do hyperparameters affect Urdu text generation quality?**
   Simplicity (1 Layer) and **Regularization** (High Dropout 0.5) were key. Increasing sequence length to 30 also improved semantic coherence.

5. **Q5: What are the failure modes of each model?**
   Transformers failed to converge due to data scarcity. RNNs suffered from occasional repetitive loops at low temperatures.

6. **Q6: How computationally efficient are different approaches?**
   RNNs were the most efficient. Transformers were significantly slower ($\approx$3x training time) and yielded poorer results, making them inefficient for this specific dataset size.

5.3 Answers to Sub-Questions

- **What is the training time vs performance tradeoff?**
  Interestingly, there was no tradeoff here. The fastest model (RNN) was also the best performing one.

- **Which model is best for resource-constrained environments?**
  The Single-Layer RNN. It requires the least memory and compute power while delivering the best generation quality.

- **Can we achieve good results with simpler models?**
  Yes. A single-layer RNN achieved a perplexity of 614.51, proving that complex architectures like Transformers are not required for small-scale poetry datasets.

References

[1] Elman, J. L. (1990). Finding structure in time. *Cognitive science*, 14(2), 179-211.

[2] Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8), 1735-1780.

[3] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, 30.

[4] Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980.*

[5] Ruder, S. (2016). An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747.*

[6] Jawahar, G., Sagot, B., & Seddah, D. (2019). What does BERT learn about the structure of language? *ACL 2019*, 3651.