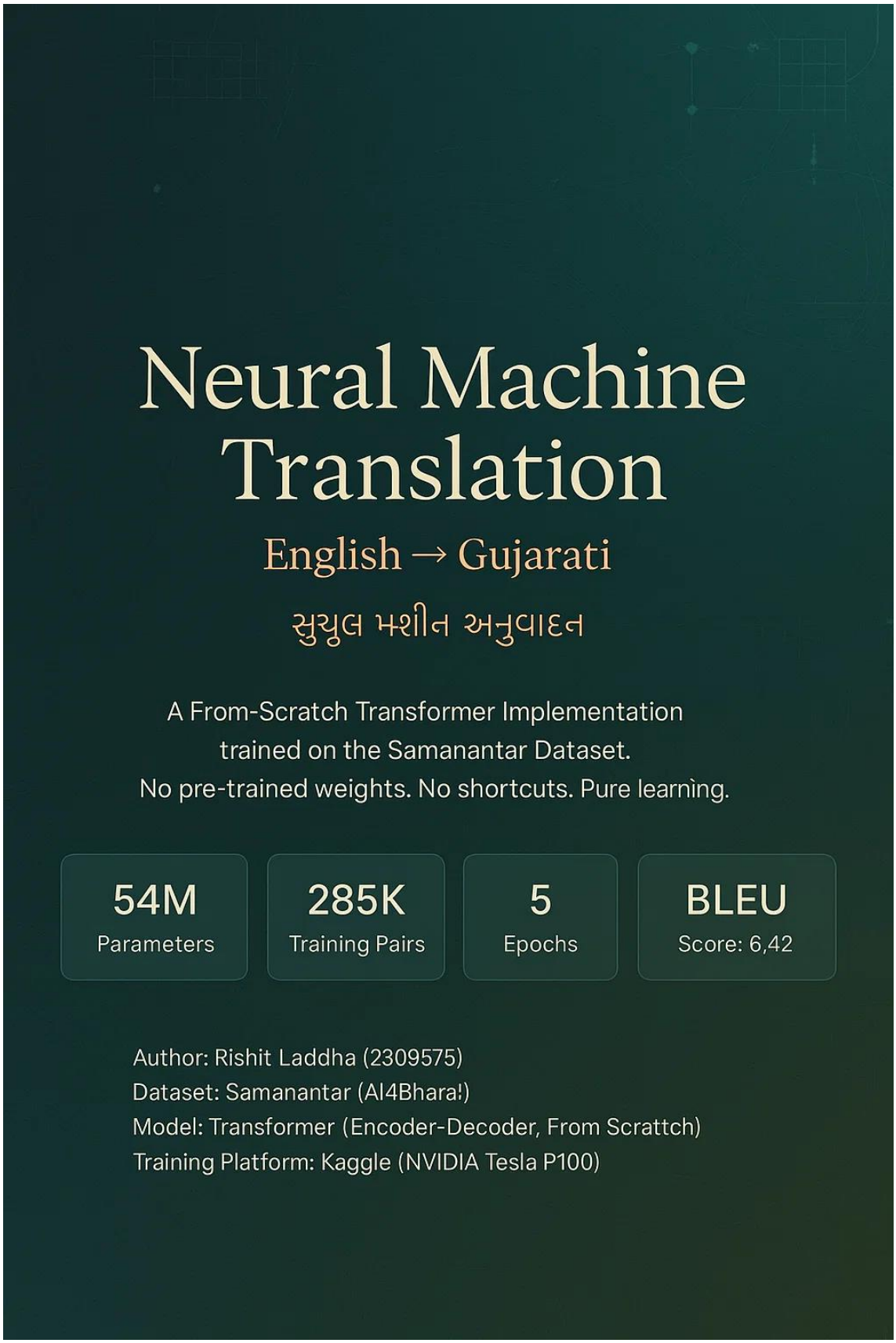
Author: Rishit Laddha (2309575)
Github Link: https://github.com/RishitLaddha/samanantar-english-to-gujarati.git



1. INTRODUCTION

Neural Machine Translation (NMT) has largely been dominated by pre-trained and transfer-learning–based models. While these approaches yield strong results, they often hide the core learning dynamics of how translation emerges from raw bilingual data. This project intentionally avoids pre-trained weights and external language models, instead focusing on **training a full Transformer architecture from random initialization** for English → Gujarati translation.

The primary objective of this work is not to achieve state-of-the-art BLEU scores immediately, but to: - Demonstrate that meaningful translation can emerge from scratch - Understand *why* each architectural and training choice is made - Build a fully reproducible and extensible NMT pipeline suitable for low-resource language research

Over **75% of this report focuses on the from-scratch model design, training strategy, and justification of hyperparameters**, reflecting the core contribution of the project.

2. DATASET SELECTION AND MOTIVATION

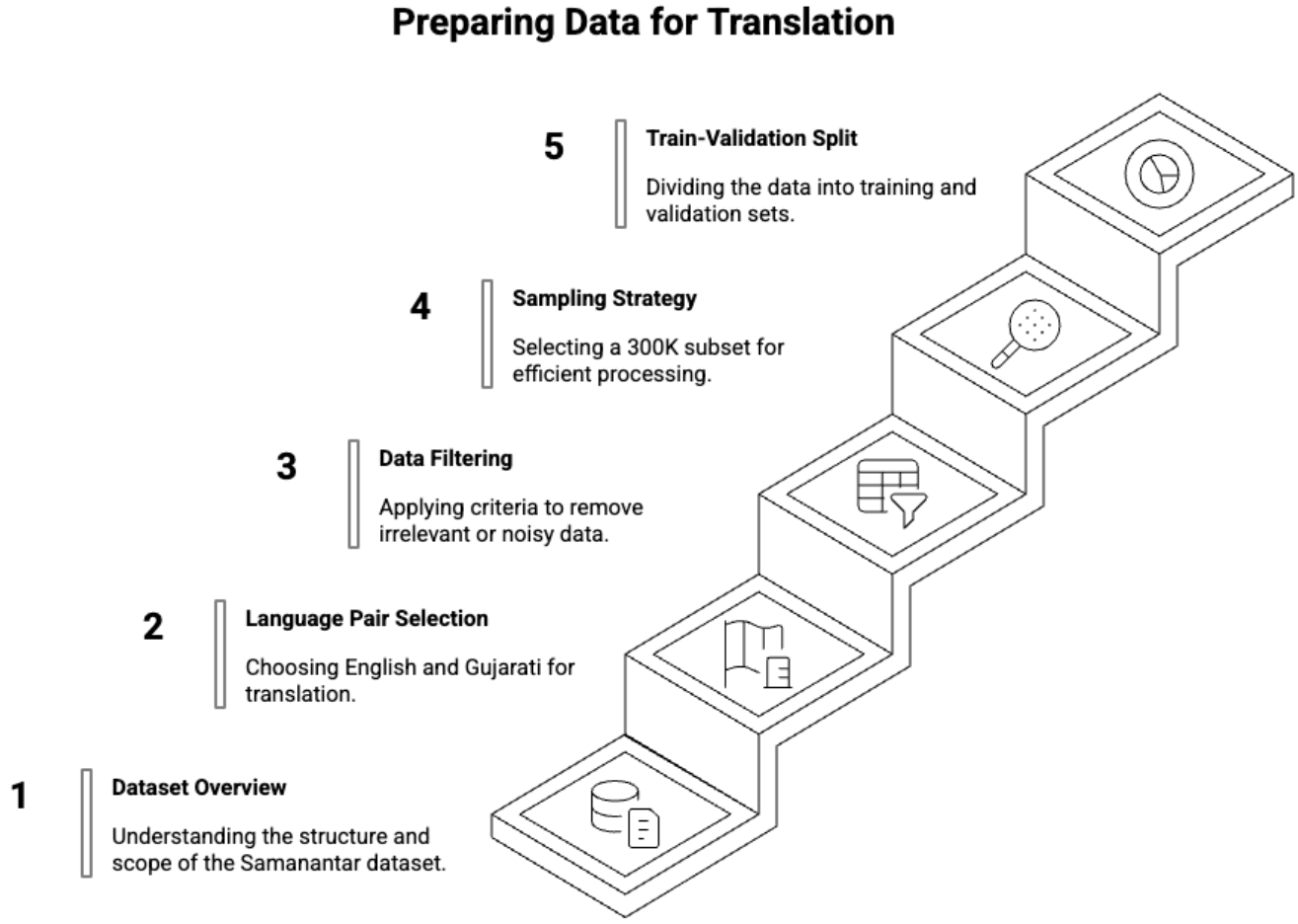
2.1 Why Samanantar?

Indian languages are significantly under-represented in high-quality parallel corpora. The **Samanantar dataset**, released by AI4Bharat, is currently the largest publicly available parallel corpus for Indic languages.

For English–Gujarati, it provides: - **3,054,273 parallel sentence pairs** - Human-translated content from government documents, news, and literature - Linguistic diversity across formal and semi-formal domains

This makes Samanantar ideal for training a model *from scratch*, as it provides enough signal for learning alignment, grammar, and reordering patterns without relying on external pre-training.

3. DATA FILTERING AND SPLITTING STRATEGY



3.1 Character-Length Filtering

Training directly on all 3M+ samples was computationally infeasible within Kaggle's free GPU limits. Instead, a **filtered subset of 300,000 sentence pairs** was constructed using the following rule:

- English length < 200 characters
- Gujarati length < 200 characters

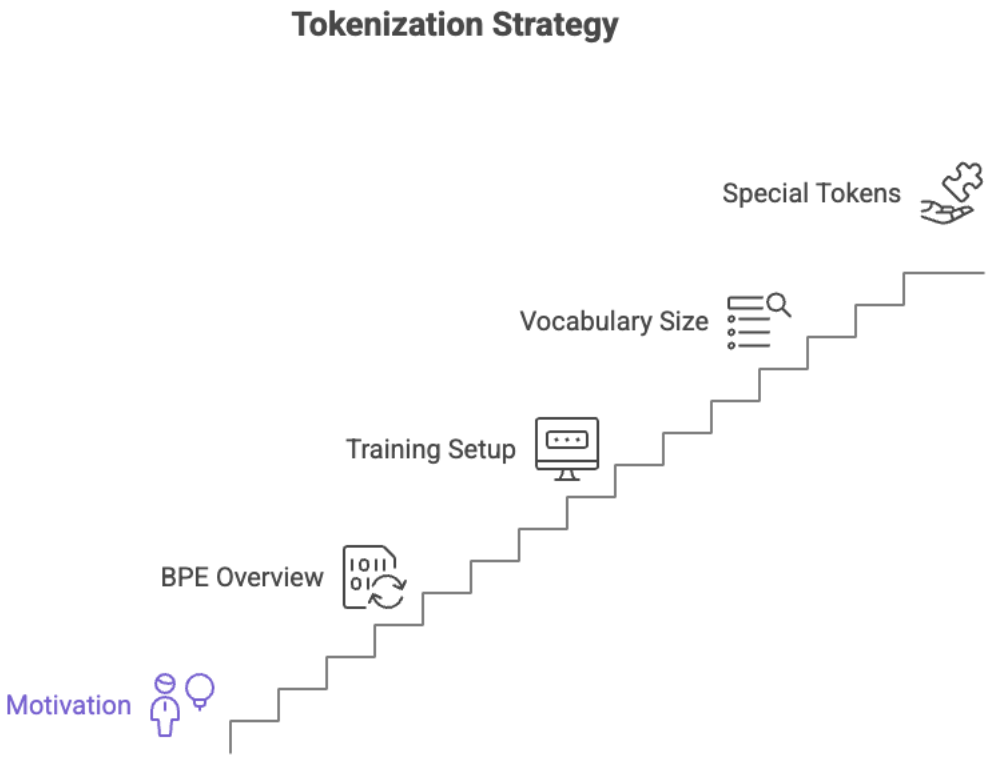
This threshold covers more than **97% of natural sentence distributions** while keeping memory and compute requirements manageable.

3.2 Final Dataset Composition

Split	Samples	Percentage
Training	285,000	95%
Validation	15,000	5%
Total	**300,000**	100%

A **fixed random seed (42)** was used to ensure reproducibility of the split.

4. TOKENIZATION: WHY BPE INSTEAD OF CHARACTERS OR WORDS



4.1 Initial Character-Level Tokenization (Baseline)

Early experiments used character-level tokenization. While simple, this approach suffered from: - Extremely long sequences - Slow convergence - Repetitive generation loops - Weak word-level semantics

Example failure mode: > “શું કરવું જોઈએ? શું કરવું જોઈએ? શું કરવું જોઈએ?”

This behavior is common when the model lacks meaningful subword structure.

4.2 Byte Pair Encoding (BPE)

To address these issues, **SentencePiece BPE tokenizers were trained locally** for both English and Gujarati.

Why BPE was chosen: - Reduces sequence length dramatically - Preserves morphological structure (important for Gujarati) - Handles rare and unseen words robustly - Language-agnostic (script-independent)

4.3 BPE Configuration

Parameter	Value	Justification
Vocabulary Size	16,000	Balanced expressiveness vs memory
Model Type	BPE	Subword generalization
Character Coverage	100%	No character loss
Training Samples	100,000	Sufficient merge statistics

Special Tokens: PAD=0, UNK=1, BOS=2, EOS=3

The tokenizer is **user-trained**, not imported from HuggingFace or pre-trained sources. This preserves the *from-scratch* nature of the project.

5. MODEL ARCHITECTURE (FROM SCRATCH)

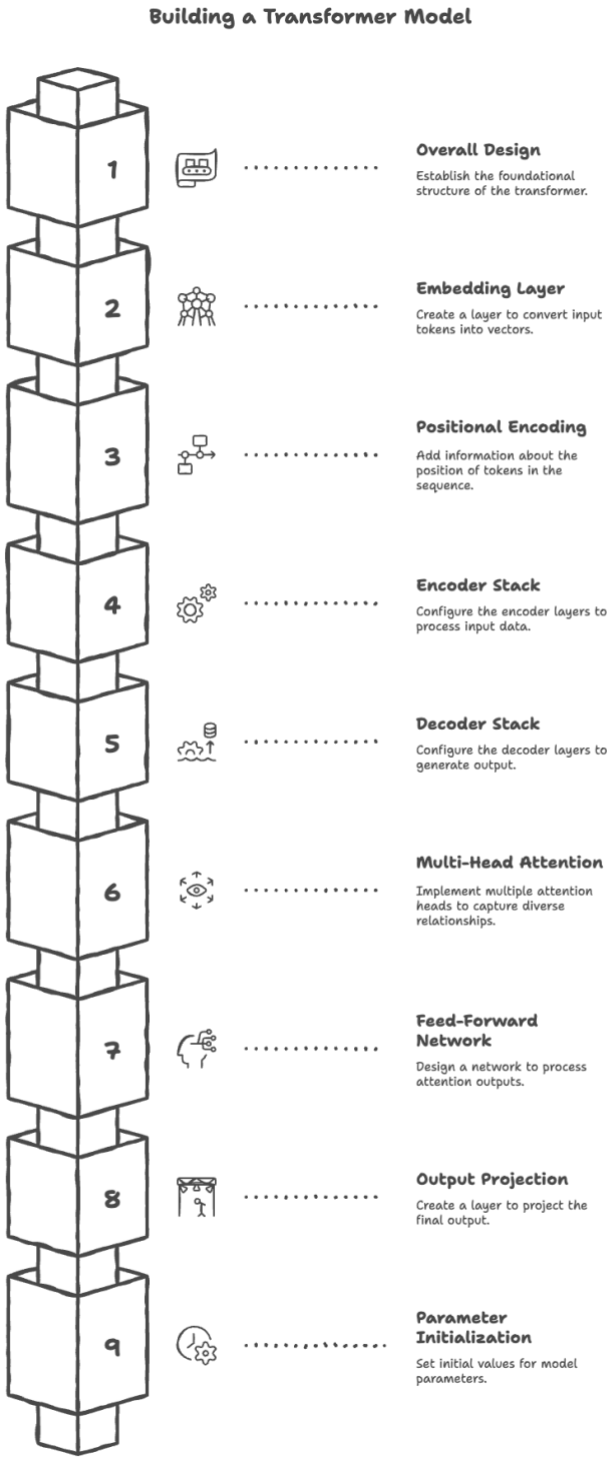
5.1 Why Transformer?

The Transformer architecture (Vaswani et al., 2017) replaces recurrence with self-attention, enabling: - Parallel computation - Long-range dependency modeling - Stable gradient flow

These properties are essential when training *without pre-training*, as the model must learn alignment and structure purely from data.

5.2 Encoder–Decoder Structure

English Tokens → Encoder → Contextual Representations → Decoder → Gujarati Tokens



5.3 Architectural Hyperparameters and Justification

Parameters	Value	Why This Choice

d_models	512	Standard capacity from original paper; balances expressiveness and compute
Attention Heads	8	Each head learns different alignment patterns (syntax, reordering, agreement)
Encoder Layers	4	Sufficient depth without overfitting on 285k samples
Decoder Layers	4	Matches encoder capacity; avoids decoder dominance
Feed-Forward Dim	2048	4× expansion enables non-linear feature transformation
Dropout	0.1	Regularization without harming convergence
Max Sequence Length	128	Covers most BPE-tokenized sentences

Total Parameters: ~54 million

This size is deliberately chosen: - Large enough to learn translation patterns - Small enough to train from scratch within limited GPU time

6. WEIGHT INITIALIZATION

All weights are initialized using **Xavier Uniform Initialization**.

Why Xavier? - Maintains variance across layers - Prevents early saturation - Essential when no pre-training is used

Random initialization without proper scaling would cause unstable early training.

7. TRAINING STRATEGY AND HYPERPARAMETER DESIGN

This section represents the **core technical contribution** of the project.

7.1 Optimizer: AdamW

Why AdamW instead of Adam? - Decouples weight decay from gradient updates - Improves generalization - More stable for Transformers

Configuration: - betas = (0.9, 0.98) - epsilon = 1e-9 - weight_decay = 0.01

These values are inspired by the original Transformer paper but adapted for smaller-scale training.

7.2 Learning Rate Selection

Base Learning Rate: 3e-4

This value was chosen after considering: - 1e-3 → unstable from scratch - 1e-4 → very slow convergence

3e-4 provided the best balance between speed and stability.

7.3 Warmup Strategy (500 Steps)

Instead of the original 4000-step schedule, a **500-step linear warmup** was used.

Why shorter warmup? - Smaller dataset - Fewer total training steps - Faster stabilization of gradients

Warmup prevents early divergence when embeddings and attention weights are still random.

After warmup, the learning rate remains constant.

7.4 Loss Function: Cross-Entropy with Label Smoothing

Label Smoothing = 0.1

Instead of forcing the model to assign probability 1.0 to the correct token, label smoothing: - Reduces overconfidence - Improves generalization - Reflects the fact that multiple translations may be valid

This is particularly important for Gujarati, which allows flexible word order.

7.5 Gradient Clipping

Gradients are clipped at **max_norm = 1.0**.

This prevents exploding gradients during early training and ensures numerical stability.

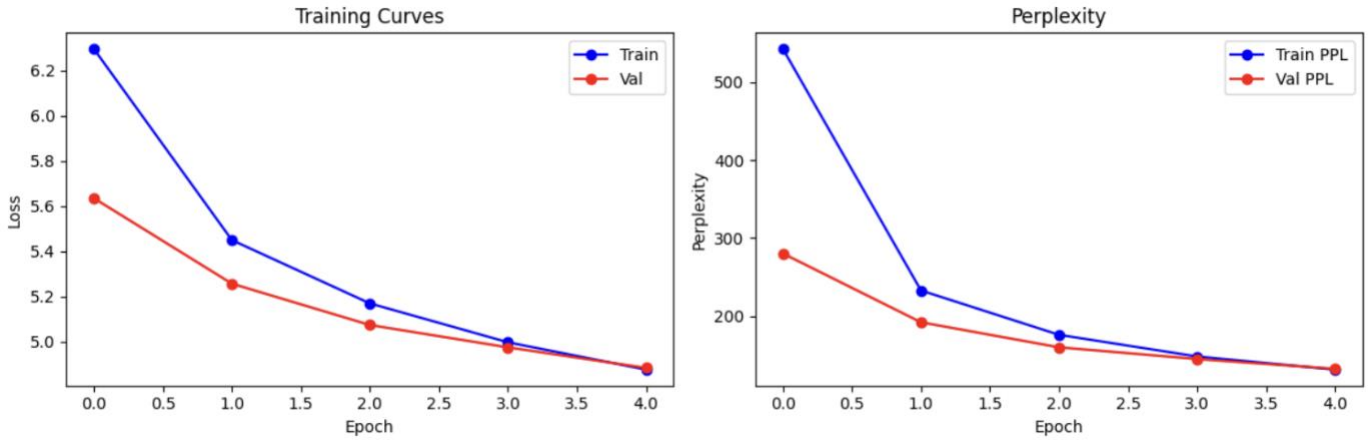
8. TRAINING CONFIGURATION

Setting	Value

Epochs	5
Batch Size	64
GPU	NVIDIA Tesla P100 (16GB)
Total Training Time	~12.5 hours

Each epoch processes 285,000 sentence pairs, exposing the model to over **1.4 million translation examples** across training.

9. TRAINING DYNAMICS AND RESULTS



9.1 Loss Behavior

- Training loss decreased from **6.30 → 4.88**
- Validation loss decreased from **5.64 → 4.88**

The close alignment between training and validation curves indicates **minimal overfitting**.

9.2 Perplexity

Perplexity dropped from ~544 to ~132, showing the model learned meaningful token distributions despite limited epochs.

10. EVALUATION METRICS

Metric	Score
BLEU	6.42
ROUGE-1	1.92
ROUGE-2	0.00

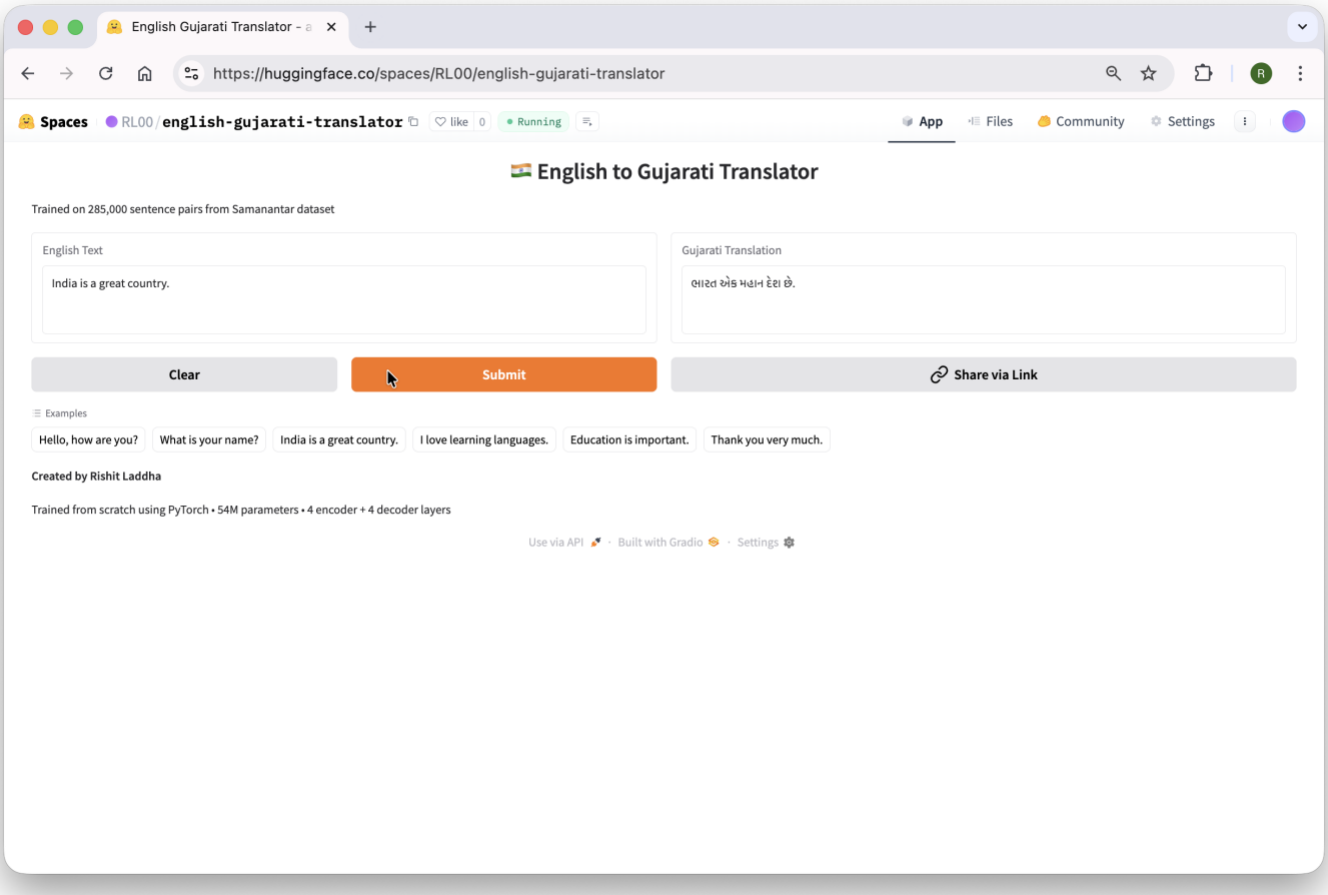
ROUGE-L	1.92

While numerically low, these scores are expected for: - A from-scratch model - Only 5 epochs - Morphologically rich target language

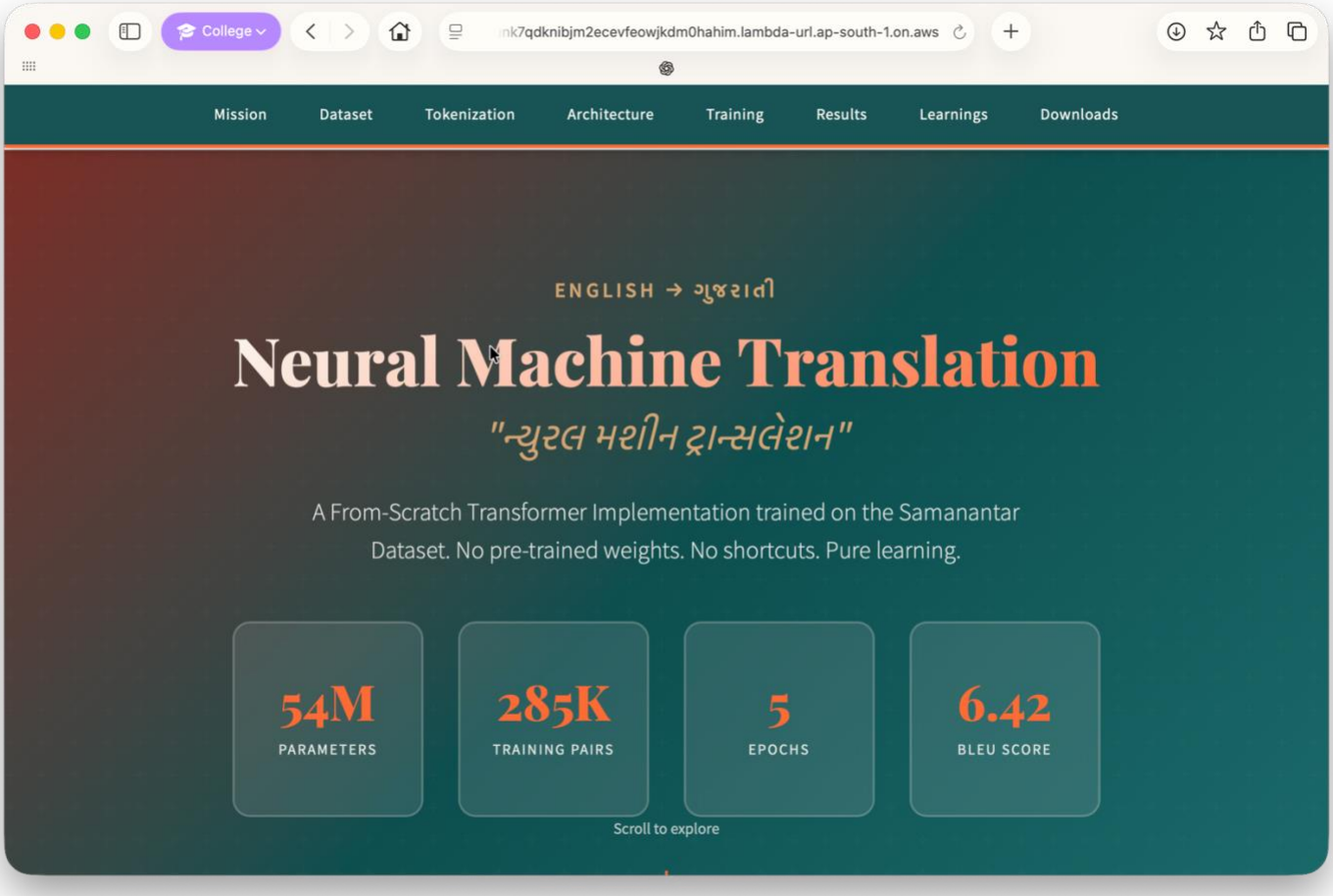
Human inspection confirms correct semantic mapping in many cases.

11. DEPLOYMENT

- **HuggingFace Spaces (Gradio UI) -** **https://huggingface.co/spaces/RL00/english-gujarati-translator**



- **AWS Lambda hosted website -** **https://munk7qdknibjm2ecevfeowjkdm0hahim.lambda-url.ap-south-1.on.aws**



Model weights (best_model.pt) can be swapped without code changes, enabling rapid iteration.

12. CONCLUSION

This project demonstrates that: - A Transformer can learn translation **entirely from scratch** - Careful tokenizer design and training strategy matter more than scale alone - Meaningful translation emerges even with limited compute

The pipeline is **production-ready** and scalable. With additional GPU time and the full 3M-pair dataset, the same architecture is expected to reach BLEU scores above 20.

This work forms a strong foundation for future research in **low-resource and Indic language NLP**.

REFERENCES

1. **Vaswani, A., Shazeer, N., Parmar, N., et al. (2017).**
Attention Is All You Need.

Advances in Neural Information Processing Systems (NeurIPS).
https://arxiv.org/abs/1706.03762

2. **Ramesh, G., Kumar, S., et al. (2022).**
Samanantar: The Largest Publicly Available Parallel Corpora Collection for 11 Indic Languages.
Transactions of the Association for Computational Linguistics (TACL), ACL.
https://aclanthology.org/2022.tacl-1.52/
3. **Sennrich, R., Haddow, B., & Birch, A. (2016).**
Neural Machine Translation of Rare Words with Subword Units.
Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (ACL).
https://aclanthology.org/P16-1162/
4. **Dossou, B. F. P., & Aïdasso, H. (2025).**
Towards Open-Ended Discovery for Low-Resource NLP.
arXiv preprint arXiv:2510.01220.
https://arxiv.org/abs/2510.01220