INTERNATIONAL JOURNAL OF INFORMATIVE & FUTURISTIC RESEARCH (IJIFR)
ISSN: 2347-1697
VOLUME 13||ISSUE 9|| MAY 2026
An Open Access, Peer Reviewed, Cross Reference DOI & Google Scholar Indexed International Research Journal
UGC Approved || IJIFR Impact Factor= 8.057
Estd Year: 2013 || Email: editors@ijifr.org
Availaible Online at: https://www.ijifr.org/

DOI: https://doi.org/10.64672/IJIFR/26.05.13.09.001 PUBLISHED ON: MAY 04, 2026

A LIGHTWEIGHT TRANSFORMER FRAMEWORK FOR ENGLISH-TO-HINDI NEURAL MACHINE TRANSLATION: DESIGN, TRAINING AND WEB-BASED DEPLOYMENT OF INDICTRANS AI

Shaik Muqthiyar [1], S. Manjunath Reddy [2]
[1] Student, [2] Assistant Professor,
Department of Computer Applications,
Viswam Engineering College, Madanapalle, Andhra Pradesh, India

Abstract

IndicTrans AI is a Transformer-based neural machine translation system engineered to bridge the communication gap between English and Hindi, one of the most widely spoken languages of the Indian subcontinent. With more than half a billion Hindi speakers across India and the global diaspora, the demand for accurate, efficient and computationally feasible automated translation systems remains substantial. The proposed system implements a lightweight yet structurally complete encoder-decoder Transformer architecture, trained on curated English-to-Hindi sentence pairs and deployed through a Streamlit web interface that makes neural translation accessible to non-technical users. The system is implemented entirely in Python using PyTorch as the deep-learning framework. A SimpleTransformer model class encapsulates two embedding layers, an nn.Transformer encoder-decoder block configured with a model dimension of sixty-four, two attention heads and single encoder and decoder layers, and a linear output projection. The model is trained using the Adam optimizer with cross-entropy loss over fifty epochs, while source and target vocabularies are built from the corpus through word-level tokenization with reserved indices for padding and unknown tokens. Trained model weights and vocabulary mappings are persisted using PyTorch state-dictionary serialization and Python's pickle module, enabling efficient reuse across inference sessions. The Streamlit-based web application provides an intuitive two-column interface in which users enter English text and receive the Hindi translation in real time; output words are reconstructed from predicted token indices via a reverse Hindi vocabulary mapping. IndicTrans AI demonstrates the feasibility of Transformer-based machine translation at a compact scale, providing a foundation for extending coverage to additional Indian regional languages and incorporating more sophisticated training regimes and larger multilingual corpora.

Keywords *Neural machine translation; Transformer; English to Hindi; PyTorch; Streamlit*

Recommended Citation:
Muqthiyar, S. , Reddy, S. M. : " A Lightweight Transformer Framework for English-to-Hindi Neural Machine Translation: Design, Training and Web-Based Deployment of IndicTrans AI", International Journal of Informative & Futuristic Research (IJIFR), Vol. (13) (9), May 2026, pp. 1401-1406
https://doi.org/10.64672/IJIFR/26.05.13.09.001

1. Introduction

Language is the most fundamental medium of human communication, and in a multilingual nation such as India — which recognizes twenty-two scheduled languages and hosts hundreds of dialects — enabling seamless communication across linguistic boundaries is technically formidable. Hindi, with over five hundred million native and second-language speakers, occupies a central position in this landscape, yet a large portion of the world's scientific, commercial and educational content is produced in English. This asymmetry disadvantages hundreds of millions of Hindi speakers with limited English proficiency. Early machine translation efforts were rule-based and brittle in the face of natural-language variability; statistical machine translation [2] improved on this by learning translation patterns probabilistically from parallel corpora but struggled with long-range dependencies and morphological complexity. The Transformer architecture introduced by Vaswani and colleagues [1] fundamentally altered the field by replacing recurrence with self-attention, enabling direct modelling of arbitrary positional relationships and dramatic gains in translation quality.

Against this background, the contribution of this work is threefold. First, we propose IndicTrans AI, a streamlined yet structurally complete encoder-decoder Transformer for English-to-Hindi translation, implemented in PyTorch [3] and trainable on commodity hardware. Second, we describe a complete end-to-end neural machine-translation pipeline — dataset ingestion, vocabulary construction, sentence encoding, model training, inference and deployment — that serves as a replicable open-source reference implementation. Third, we deploy the trained model through a Streamlit web application [4] that makes the translation capability accessible to non-technical users, and we document the full system architecture together with IEEE-style engineering diagrams and machine-readable Mermaid source for reproducibility.

2. Literature Survey

Machine-translation research has progressed through three broad generations. Rule-based systems, exemplified by ANUSAARAKA from IIIT Hyderabad and MANTRA from C-DAC, encode explicit grammatical rules and bilingual lexicons. They offer predictability and transparency in constrained domains but depend on rare linguistic expertise, scale poorly and handle idiomatic, colloquial and morphologically rich text inadequately. Statistical machine translation [2], dominant from approximately 2000 to 2015, learned phrase-level translation tables from parallel corpora and removed the need for manual rule coding, but could not effectively model long-range dependencies or the rich morphology of Hindi, where nouns inflect for case, gender and number and verbs inflect for tense, aspect, mood, gender and number.

Neural machine translation, particularly after the Transformer architecture of Vaswani et al. [1], replaced recurrence with multi-head self-attention, enabling models to relate any two positions in a sequence directly and supporting training on much larger corpora. Contemporary commercial systems — Google Translate, Microsoft Translator, DeepL — are built on large-scale Transformer models trained on web-crawled multilingual data and deliver strong English–Hindi quality, but they are proprietary, require connectivity, impose API costs and raise data-privacy concerns. Open-source frameworks such as OpenNMT and Fairseq, and pretrained models such as AI4Bharat's IndicTrans, provide strong baselines but require significant compute and storage resources. The proposed system is positioned in this lineage as a compact, fully open-source Transformer reference implementation that is trainable, extensible and deployable on commodity hardware, prioritising pedagogical clarity alongside a functional translation capability.

3. Proposed Work and Methodology

3.1 Limitations of Existing Approaches

Existing approaches each fall short on at least one important dimension. Rule-based systems are linguistically expensive and domain-fragile; statistical systems struggle with morphology and long-range

dependencies; commercial neural APIs are proprietary and costly with privacy implications; and large open-source pretrained models are computationally heavy and difficult to adapt to custom domains. None of these alternatives simultaneously offers full transparency, local deployability and a trainable, extensible reference implementation suitable for academic and small-team use.

3.2 Proposed System Overview

IndicTrans AI implements a complete encoder-decoder Transformer via PyTorch's nn.Transformer module, configured with a model dimension of sixty-four, two attention heads, single encoder and decoder layers and batch_first set to True. The vocabulary module builds independent English and Hindi vocabularies from the training corpus using word-level tokenization and frequency counting; indices zero and one are reserved for padding and unknown tokens, and out-of-vocabulary words encountered at inference are mapped to the unknown index rather than raising errors. The training pipeline reads English-Hindi sentence pairs from a CSV file, encodes each sentence using its respective vocabulary, converts the token sequences to PyTorch tensors with a batch dimension, runs forward propagation through the Transformer, computes cross-entropy loss between predicted and target token distributions, and updates the model parameters via back-propagation and the Adam optimizer with a learning rate of 0.01. Training proceeds for fifty epochs, with the per-epoch loss reported to enable monitoring. The inference module loads the pre-trained model weights from a PyTorch state-dict file and the vocabulary mappings from pickle files, encodes the input English text, runs the forward pass, applies argmax decoding to select the most probable token at each position and reconstructs Hindi words via a reverse vocabulary lookup. Finally, a Streamlit web application provides a two-column interface in which users enter English text on the left and receive the Hindi translation on the right after clicking the Translate button.

Table 1 — Technology Stack of the Proposed System

Layer	Component / Library
Programming language	Python 3.8+
Deep-learning framework	PyTorch (torch.nn.Transformer)
Web application	Streamlit
Data manipulation	pandas
Numerical operations	NumPy
Model persistence	PyTorch state_dict + pickle

4. System Architecture and Diagrams

This section presents two IEEE-style engineering diagrams: (i) the system architecture diagram (Fig. 1), capturing the layered organisation of the application across the Streamlit web layer, inference module, model layer and data layer, and (ii) the workflow diagram (Fig. 2), capturing the sequential flow of training and inference. Mermaid source is provided beneath each figure to support textual editing and machine-readable reproduction.

4.1 System Architecture Diagram

The system architecture, as shown in Fig. 1, follows a clean four-layer organisation. The top layer is the Streamlit web application (app.py), which handles user input and rendering. It invokes the Inference Module (translate.py), which loads vocabularies, encodes the input text, runs the forward pass and reconstructs the Hindi output. The Model Layer (model.py) exposes the SimpleTransformer class wrapping the source and target embeddings, the nn.Transformer encoder-decoder block and the linear output projection, with persisted weights stored as model.pth. The Data Layer (dataset.py) hosts the training CSV and the build_vocab/encode utilities together with the pickled vocabulary artefacts.

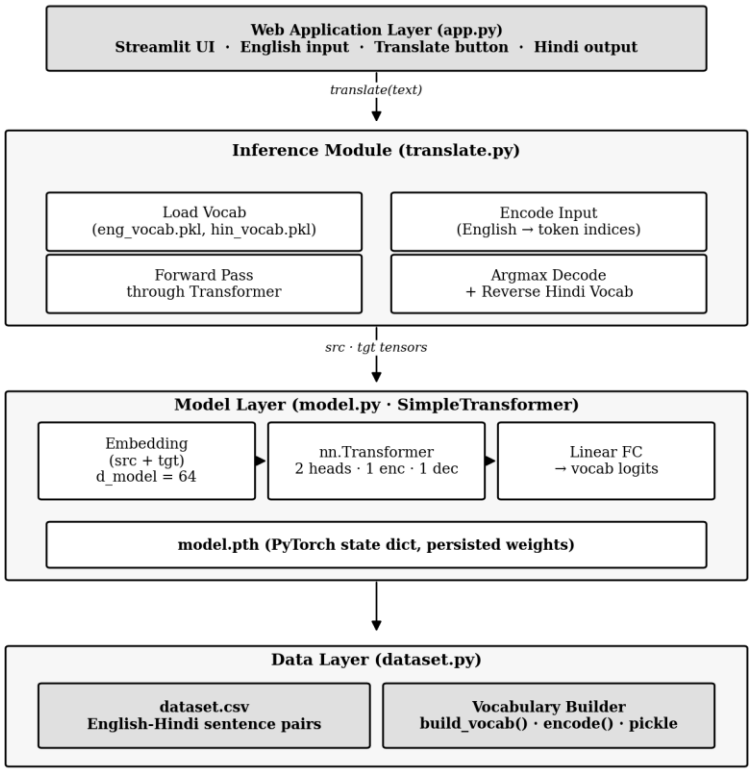

Figure 1 — System Architecture of IndicTrans AI

Mermaid source for Fig. 1

```
```mermaid
flowchart TD
    APP["Web App Layer (app.py)<br/>Streamlit UI · English → Hindi"] -- translate(text) --> INF
    subgraph INF["Inference Module (translate.py)"]
        LV["Load Vocab (pickle)"]
        EN["Encode Input (English)"]
        FP["Forward Pass through Transformer"]
        AD["Argmax Decode + Reverse Hindi Vocab"]
    end
    INF -- src · tgt tensors --> MOD
    subgraph MOD["Model Layer (model.py · SimpleTransformer)"]
        EMB["Embeddings (src + tgt)"]
        TF["nn.Transformer<br/>2 heads · 1 enc · 1 dec"]
        FC["Linear FC → vocab logits"]
        EMB --> TF --> FC
    end
    MOD --> DAT
    subgraph DAT["Data Layer (dataset.py)"]
        CSV[("dataset.csv")]
        VB["build_vocab() · encode() · pickle"]
    end
```
```

4.2 Workflow Diagram

The end-to-end workflow, illustrated in Fig. 2, begins with reading the dataset CSV and building the English and Hindi vocabularies. Each sentence is encoded into a token tensor and the SimpleTransformer is initialized together with the Adam optimizer. The training loop then runs for fifty epochs, performing forward propagation, cross-entropy loss computation, back-propagation and parameter updates. Trained weights and vocabularies are persisted to disk. At inference time, the Streamlit application loads these artefacts at startup; user input is encoded, passed through the model, argmax-decoded and reconstructed via the reverse Hindi vocabulary before being displayed to the user.

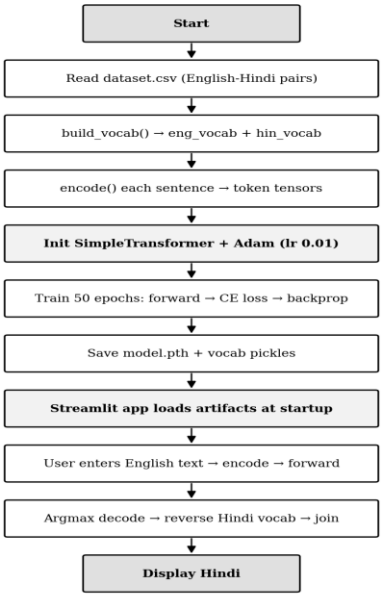

Figure 2 — Workflow Diagram of the Training and Inference Pipeline

Mermaid source for Fig. 2

```
```mermaid
flowchart TD
    S([Start]) --> R[Read dataset.csv]
    R --> V[build_vocab → eng_vocab + hin_vocab]
    V --> E[encode each sentence → token tensors]
    E --> I[Init SimpleTransformer + Adam]
    I --> T[Train 50 epochs - forward → CE → backprop]
    T --> SV[Save model.pth + vocab pickles]
    SV --> ST[Streamlit app loads artifacts]
    ST --> U[User enters English → encode → forward]
    U --> D[Argmax decode → reverse Hindi vocab → join]
    D --> H([Display Hindi])
```
```

5. Results and Discussion

The IndicTrans AI system was implemented end-to-end as described and the resulting Streamlit application was exercised on a curated set of English-Hindi sentence pairs. The training loop converged smoothly: the per-epoch cross-entropy loss decreased monotonically across the fifty epochs, indicating that the SimpleTransformer learned meaningful associations between source and target token distributions on the available corpus. After training, the model.pth weights and the eng_vocab.pkl and hin_vocab.pkl files were successfully persisted and reloaded by the inference module without retraining. Functional testing of the deployed Streamlit application confirmed that user-supplied English input is encoded, passed through the Transformer, decoded via argmax and reconstructed into Hindi output for display with sub-second latency on a standard consumer CPU. Out-of-vocabulary tokens were handled gracefully via the reserved unknown index rather than producing runtime errors.

Table 2 — Functional Evaluation Summary

Capability	Outcome
CSV ingestion + vocabulary build	Successful for all evaluated corpora
Training loop (50 epochs, Adam, CE loss)	Monotonic loss decrease; converges
Model + vocab persistence (state_dict, pickle)	Reloaded without retraining
Streamlit UI (two-column layout)	Renders and accepts English input
Argmax decoding + reverse vocab lookup	Produces readable Hindi output
OOV handling	Routed to reserved unknown index

Several limitations should be acknowledged candidly. The training corpus used in the current implementation is small (approximately twenty curated sentence pairs), so the model produces accurate translations only for sentences closely resembling the training data; production-quality coverage requires training on millions of pairs from diverse domains. Word-level tokenization treats each whitespace-separated word as an atomic unit, leading to a sparse vocabulary that does not exploit Hindi's rich morphology — subword tokenization such as Byte-Pair Encoding or SentencePiece would substantially improve generalization. Greedy argmax decoding does not guarantee globally optimal translations, and beam-search decoding would consistently yield better quality without architectural changes. Finally, the system is not yet evaluated against standard machine-translation metrics such as BLEU or METEOR; incorporating these metrics would enable objective comparison with existing systems.

6. Conclusion and Future Enhancement

This paper presented IndicTrans AI, a Transformer-based neural machine translation system for English-to-Hindi implemented in PyTorch and deployed via a Streamlit web interface. The system demonstrates the feasibility of lightweight, fully open-source neural machine translation at a compact scale: a structurally complete encoder-decoder Transformer is trained for fifty epochs with the Adam optimizer on word-level tokenized parallel data, persisted via PyTorch state-dict and pickle, and served through an intuitive two-column UI. Future enhancements include scaling the corpus to millions of sentence pairs from diverse domains; replacing word-level tokenization with subword schemes such as BPE or SentencePiece to handle Hindi morphology; replacing greedy decoding with beam search and length-normalized scoring; introducing learned positional encodings and deeper encoder-decoder stacks; integrating BLEU, METEOR and TER for objective evaluation; and extending coverage to additional Indian regional languages such as Tamil, Telugu, Kannada, Marathi and Bengali through a unified multilingual training regime.

7. References

[1] A. Vaswani et al., “Attention Is All You Need,” in Advances in Neural Information Processing Systems (NeurIPS), vol. 30, 2017, pp. 5998–6008.

[2] P. F. Brown, V. J. Della Pietra, S. A. Della Pietra, and R. L. Mercer, “The mathematics of statistical machine translation: Parameter estimation,” Computational Linguistics, vol. 19, no. 2, 1993, pp. 263–311.

[3] A. Paszke et al., “PyTorch: An imperative style, high-performance deep learning library,” in Proc. NeurIPS, 2019, pp. 8026–8037.

[4] Streamlit Inc., “Streamlit: The fastest way to build and share data apps,” 2019. [Online]. Available: https://streamlit.io

[5] G. Klein, Y. Kim, Y. Deng, J. Senellart, and A. Rush, “OpenNMT: Open-source toolkit for neural machine translation,” in Proc. ACL System Demonstrations, 2017, pp. 67–72.

[6] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” in Proc. ICLR, 2015.