Designing a Multilingual Translation & Transliteration System

Problem, Data, Cleaning, Models, and Inference – End to End

Chandan J

July 23, 2025

Agenda

What Are We Building?

Goal

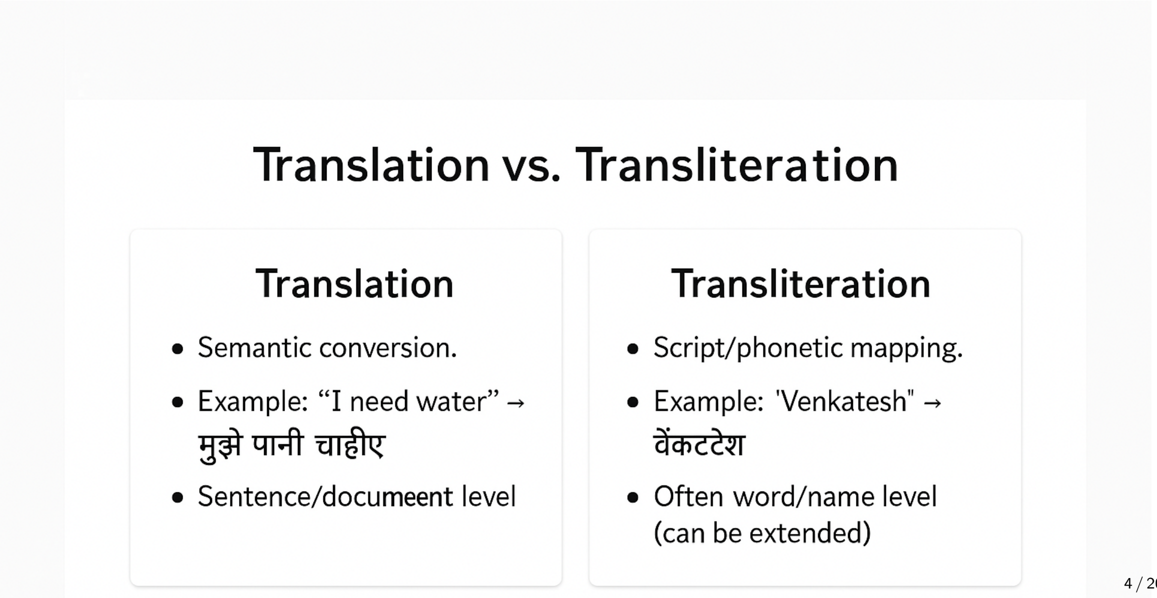
Design an AI system that can:

- **Translate**: Convert meaning from a source language to a target language.
- **Transliterate**: Convert the script/phonetics of text while keeping pronunciation (e.g., Romanized Hindi to Devanagari).

Key Decisions

- Language pairs and directions (uni- vs. bi-directional).
- Real-time vs. batch usage; cloud vs. edge deployment.
- Quality vs. latency trade-offs (beam width, model size, quantization).

Translation vs. Transliteration



Parallel Translation Corpora

Dataset	**Langs**	**Domain**	**Size (pairs)**
FLORES-200	200	Balanced eval	$\sim$3k/dev & test/lang
OPUS (TED, Tatoeba, etc.)	400+	Varied (talks, subtitles)	10k–Millions
WMT (News/ParaCrawl)	10–30	News/Web	Millions
Samanantar (Indic)	11 Indic-En	Web, curated	$\sim$49M
IITB En-Hi	En-Hi	Mixed (tech/news)	$\sim$1.5M
PMIndia	13 Indic-En	Govt. releases	$\sim$0.5M

Transliteration Datasets

Dataset	Langs/Scripts	Task	Size
Dakshina	12 Indic	Roman $\leftrightarrow$ Native	$\sim$1.7M pairs
Aksharantar	20 Indic	Name/word translit.	$\sim$4.7M pairs
NEWS Shared Task	15–20	Name transliteration	20k–100k/lang
WikiTitles/WikiNames	100+	Entity translit.	Millions (noisy)

Monolingual & Code-Mixed Corpora

Monolingual: mC4, OSCAR, IndicCorp, Wikipedia, News Crawl (useful for back-translation and LM pretraining).

Code-mixed/Romanized: LinCE, GLUECoS, FIRE Hinglish, CMU Hinglish datasets.

Why needed?

- Augment low-resource languages via back-translation.
- Robustness to spelling/code-mixing variations.

Cleaning Pipeline

1. **Normalize Unicode** (NFC/NFKC), remove control characters.
2. **Deduplicate** sentence pairs and drop near-identical duplicates.
3. **Length filters**: remove pairs with extreme length ratios (e.g., $\geq$ 3:1) or too long/short sentences.
4. **Language ID (LID) filtering**: ensure each side is actually in the declared language.
5. **Punctuation & whitespace normalization**.
6. **Script normalization** for Indic languages (e.g., nukta forms).
7. **Remove noisy HTML, emojis (if not needed)**.

Tokenization & Vocab

Translation

Subword methods: SentencePiece (BPE / unigram LM) with joint vocabulary.

Typical vocab size: 16k–64k.

Special language tags: `<2hi>` to control target language.

Transliteration

Character-level tokenization often sufficient.

Explicitly control allowed output character set per language.

Model Options: Translation

Baselines

- Seq2Seq LSTM + Attention
- Transformer (Encoder–Decoder)

Advanced

- Pretrained mBART, mT5, NLLB fine-tuning
- Mixture-of-Experts for many languages
- Shared encoder/decoder with language embeddings

Model Options: Transliteration

Char-level Seq2Seq with attention (BiLSTM encoder, LSTM decoder).

Transformer at character level (lighter, fast to train).

Rule-based / WFST baseline for comparison and error analysis.

Pronunciation-aware (G2P, phoneme embeddings) if speech-oriented.

Typical Training Loop (PyTorch Sketch)

```
for epoch in range(num_epochs):
    model.train()
    for batch in train_loader:
        src_ids, src_mask, tgt_in, tgt_out, tgt_mask = batch
        logits = model(src_ids, src_mask, tgt_in, tgt_mask)
        loss   = label_smoothing_ce(logits, tgt_out, eps=0.1)

        loss.backward()
        torch.nn.utils.clip_grad_norm_(model.parameters(), 1.0)
        optimizer.step(); optimizer.zero_grad()

    valid_score = validate(model, valid_loader)
    scheduler.step()
    save_checkpoint(...)
```

Training Tips

Label smoothing (e.g., $\epsilon = 0.1$) improves generalization.

Mixed precision (FP16) for speed and memory.

Gradient accumulation for large batches on small GPUs.

Early stopping based on BLEU/chrF (translation) or CER (transliteration).

Track experiment configs with YAML + DVC/Weights&Biases.

Decoding / Inference

Beam Search (Translation)

Beam size 4–8 usually a good trade-off.

Length penalties to avoid too short outputs.

Constrained decoding (e.g., glossary constraints) if needed.

Constrained Output (Transliteration)

Restrict output to valid target script characters.

Use top-k candidates (k=5) for ambiguous names; rank by language model.

Post-processing

Detokenization / script re-normalization.

Recasing and punctuation restoration if tokenization was case-insensitive.

Handling unknown tokens with copy-mechanisms or dictionary lookups.

For transliteration: unify diacritics, remove duplicates.

Metrics

Translation

- BLEU, chrF, TER
- COMET / BERTScore (semantic)
- Human eval: Adequacy & Fluency, MQM

Transliteration

- Exact Match Accuracy
- Character Error Rate (CER) / Levenshtein distance
- Top-k accuracy

Serving the Model

Export: TorchScript / ONNX $\rightarrow$ TensorRT for low latency.

REST/gRPC API (FastAPI $+$ Uvicorn).

Quantization (INT8), pruning for edge devices.

Batch vs. streaming inference depending on use-case.

MLOps & Monitoring

CI/CD: unit tests for tokenizers, data loaders, decoding.

Version checkpoints & data (Git-LFS / DVC).

Monitor drift in input language distribution.

Feedback loop: collect user corrections for continual fine-tuning.

Takeaways

Choose clear scope: translation vs. transliteration (or both).

Curate and clean robust datasets (parallel, monolingual, code-mixed).

Baseline first (Seq2Seq/Transformer), then scale (pretrained, MoE).

Carefully design inference (beam, constraints) and evaluation (BLEU/CER).

Plan for deployment, monitoring, and continuous improvement.

Questions?