

# Encoder–Decoder Models for Machine Translation: Detailed Manual

Expanded from lecture slides

## Contents

<b>1</b>	<b>High-level goal and problem statement</b>	<b>2</b>
<b>2</b>	<b>Encoder–Decoder architecture (intuitive)</b>	<b>2</b>
2.1	Why encoder–decoder? . . . . .	2
2.2	Components . . . . .	2
2.3	Training objective . . . . .	2
<b>3</b>	<b>Encoder: details and shapes</b>	<b>2</b>
<b>4</b>	<b>Decoder: detailed anatomy</b>	<b>2</b>
<b>5</b>	<b>Cross-attention intuition</b>	<b>3</b>
<b>6</b>	<b>Decoding at inference</b>	<b>3</b>
6.1	Greedy decoding . . . . .	3
6.2	Beam search (standard for MT) . . . . .	3
6.2.1	Scoring and length normalization . . . . .	4
6.2.2	Beam-search pitfalls . . . . .	4
<b>7</b>	<b>Training details and data sources</b>	<b>4</b>
7.1	Parallel corpora (bitext) . . . . .	4
7.2	Loss function recap . . . . .	4
<b>8</b>	<b>Advanced topics and practical techniques</b>	<b>4</b>
8.1	Backtranslation (data augmentation) . . . . .	4
8.2	Multilingual training . . . . .	5
8.3	Vocabulary and subword segmentation . . . . .	5
8.4	Evaluation metrics . . . . .	5
<b>9</b>	<b>Algorithmic and computational considerations</b>	<b>5</b>
9.1	Inference efficiency . . . . .	5
9.2	Memory shapes for KV cache . . . . .	5
<b>10</b>	<b>Beam search: worked toy example</b>	<b>5</b>
<b>11</b>	<b>Practical tips and common gotchas</b>	<b>5</b>
<b>12</b>	<b>Summary checklist</b>	<b>6</b>

# 1 High-level goal and problem statement

**Machine Translation (MT):** Given a source-language token sequence  $x = (x_1, \dots, x_n)$ , produce a target-language token sequence  $y = (y_1, \dots, y_m)$  that correctly conveys the meaning of  $x$  in the target language.

We frame MT as modeling the conditional probability

$$P(y_1, \dots, y_m \mid x_1, \dots, x_n) = \prod_{t=1}^m P(y_t \mid y_{<t}, x)$$

and learning parameters to maximize this probability on a parallel corpus (bitext).

## 2 Encoder–Decoder architecture (intuitive)

### 2.1 Why encoder–decoder?

- Input and output sequences can differ in length and order.
- The encoder produces a contextual representation of the source sentence; the decoder is a conditional language model that generates the target sentence while attending to the source representation.

### 2.2 Components

1. **Encoder:** a stack of transformer (or RNN/CNN) encoder blocks that map tokens  $x$  to a sequence of contextual vectors  $H^{enc} = (h_1, \dots, h_n)$ .
2. **Decoder:** a stack of transformer decoder blocks. Each decoder layer includes: masked self-attention (left-to-right), cross-attention to encoder outputs, and a feed-forward sublayer.
3. **Output projection / softmax:** maps decoder output at each time step to logits over the target vocabulary and applies softmax to get  $P(y_t \mid \cdot)$ .

### 2.3 Training objective

Given a parallel pair  $(x, y)$ , minimize cross-entropy loss (negative log-likelihood) across target positions:

$$\mathcal{L}(\theta) = - \sum_{t=1}^m \log P_{\theta}(y_t \mid y_{<t}, x).$$

Optimization is typically done with Adam/AdamW on large bitext.

## 3 Encoder: details and shapes

The encoder is identical to the transformer encoder used in many tasks. Input tokens  $x_i$  are embedded and summed with positional encodings to form  $X \in \mathbb{R}^{n \times d}$ . After  $L$  layers we obtain  $H^{enc} = H^{(L)} \in \mathbb{R}^{n \times d}$  where each row  $h_j$  summarizes the source token  $x_j$  in context.

## 4 Decoder: detailed anatomy

Each decoder layer  $\ell$  consists of three sublayers (in order):

1. Masked multi-head self-attention (prevents attending to future target positions).

2. Cross-attention (multi-head) where queries come from decoder previous layer and keys/values come from encoder outputs  $H^{enc}$ .
3. Position-wise feed-forward network (FFN).

Mathematically, for decoder input  $S^{(\ell-1)} \in \mathbb{R}^{t \times d}$  ( $t$  = current target length during training), cross-attention computes:

$$Q = S^{(\ell-1)} W_Q^{(cross)}, \quad K = H^{enc} W_K^{(cross)}, \quad V = H^{enc} W_V^{(cross)},$$

and

$$\text{CrossAtt}(Q, K, V) = \text{softmax} \left( \frac{QK^\top}{\sqrt{d_k}} \right) V.$$

This produces a tensor of shape  $(t \times d)$  that mixes encoder information into the decoder states.

### Shapes summary

- Encoder outputs:  $H^{enc} \in \mathbb{R}^{n \times d}$ .
- Decoder queries (cross-attention):  $Q \in \mathbb{R}^{t \times d_k}$ .
- Keys/Values:  $K, V \in \mathbb{R}^{n \times d_k}$  (per head) or  $(n \times d)$  after concatenation.

## 5 Cross-attention intuition

Cross-attention lets the decoder "read" the entire source representation when generating each target token. For target position  $t$ , its query  $Q_t$  asks "what should I pay attention to in the source?" The attention weights over source positions determine which source words contribute most to generating the next target word.

## 6 Decoding at inference

### 6.1 Greedy decoding

At each step choose the most probable token:

$$y_t = \arg \max_w P(y_t = w \mid y_{<t}, x).$$

Fast but can lead to suboptimal full-sequence probability because local choice may hurt later scores.

### 6.2 Beam search (standard for MT)

Beam search maintains  $k$  hypotheses (partial sequences) at each time step. It expands each hypothesis by all vocabulary tokens, scores the new hypotheses (sum of log-probs), and prunes to top- $k$ .

Pseudocode (concise):

```
beam = [[BOS], score=0]
for t=1..max_len:
    candidates = []
    for seq, score in beam:
        probs = model.predict_next(seq, x) # log-probs over vocab
        for token, logp in topVocab(probs):
            candidates.append((seq+[token], score+logp))
    beam = topk(candidates, k)
    if all sequences ended with EOS: break
return best completed sequence by normalized score
```

### 6.2.1 Scoring and length normalization

Because log-probabilities favor shorter sequences (product of probabilities), many implementations use length normalization or a length-penalty:

$$\text{score}(y) = \frac{1}{(5 + |y|)^\alpha} \sum_{t=1}^{|y|} \log P(y_t \mid y_{<t}, x)$$

where  $\alpha$  is tuned on validation data (commonly 0.6).

### 6.2.2 Beam-search pitfalls

- Large beam sizes can sometimes degrade BLEU due to mismatch between model probability and evaluation metric.
- Beam search can favor generic safe outputs.
- Length bias: without length normalization the search prefers shorter sequences.

## 7 Training details and data sources

### 7.1 Parallel corpora (bitext)

Common corpora used in MT:

- Europarl (parliament proceedings) — many European languages.
- United Nations parallel corpus — 6 official UN languages.
- OpenSubtitles — informal movie subtitles.
- ParaCrawl — web-crawled parallel text between many EU languages and English.

### 7.2 Loss function recap

Cross-entropy at each target position:

$$\mathcal{L}_{CE}(y_{1:m}, x) = - \sum_{t=1}^m \log P(y_t \mid y_{<t}, x).$$

During training we use teacher forcing: at step  $t$  the decoder conditions on ground-truth prefix  $y_{<t}$  rather than sampled predictions. This stabilizes and accelerates training.

## 8 Advanced topics and practical techniques

### 8.1 Backtranslation (data augmentation)

- Goal: improve source-to-target MT when parallel data is scarce but monolingual target data is abundant.
- Procedure: train a reverse model (target  $\rightarrow$  source) *on available bitext*; use it to translate monolingual target sentences *target model*.
- Empirical result: backtranslation yields large gains; synthetic data often gets 2/3 of the benefit of real bitext.

## 8.2 Multilingual training

Train one model on many language pairs by adding language-id tokens:

- Prepend a source-language token  $\langle L_{src} \rangle$  to the encoder input and/or prepend a target-language token  $\langle L_{tgt} \rangle$  to decoder prompts.
- The model learns to translate between many pairs and can transfer knowledge to low-resource languages.

## 8.3 Vocabulary and subword segmentation

Use subword tokenization (BPE, WordPiece, SentencePiece) to handle rare words and morphological variation. Vocabulary choices affect coverage and length; multilingual vocabularies are larger and use shared subword units.

## 8.4 Evaluation metrics

- **BLEU**: n-gram precision with length penalty and brevity penalty; most common automatic metric.
- **chrF**: character n-gram F-score, useful for morphologically rich languages.
- **TER**: translation edit rate (lower is better).
- Human evaluation remains the gold standard for fluency/adequacy.

# 9 Algorithmic and computational considerations

## 9.1 Inference efficiency

Decoder uses auto-regressive generation: naive attention costs grow with generated length. Common optimizations:

- KV-caching: store encoder K/V and previously computed decoder K/V to avoid recomputation.
- Batch multiple beams efficiently using vectorized matmuls.

## 9.2 Memory shapes for KV cache

A cached layout often uses shape: (layers, batch\*beams, heads, seq<sub>len</sub>, d<sub>head</sub>) to support fast matmuls and batch beam search.

# 10 Beam search: worked toy example

Toy vocabulary: yes, no, ok, EOS. Suppose model on input x assigns at t=1: P(yes)=0.5, P(no)=0.3, P(ok)=0.2. With beam size k=2 we keep [yes(0.5), no(0.3)]. At t=2 expand: suppose given prefix yes, model gives EOS:0.8, no:0.2; for prefix no gives ok:0.6, EOS:0.4. Compute joint scores: yes+EOS=0.5\*0.8=0.4, yes+no=0.5\*0.2=0.1, no+ok=0.3\*0.6=0.18, no+EOS=0.3\*0.4=0.12. Top-2 sequences by probability: yes EOS (0.4), no ok (0.18). Beam search returns yes EOS as top final hypothesis even if the globally most probable 3-token sequence might have been different.

# 11 Practical tips and common gotchas

- Tune beam size and length penalty on dev set; larger beams aren't always better.

- Use diverse decoding for creative generation tasks (not typical for MT).
- When backtranslating, filter synthetic pairs by language model scores to remove noisy outputs.
- For low-resource languages, multilingual transfer and parameter sharing help.

## 12 Summary checklist

- Encoder produces source contextual vectors; decoder generates target tokens auto-regressively while cross-attending to encoder outputs.
- Train with cross-entropy on parallel corpora; use teacher forcing.
- Decode with beam search, tune length penalty; use KV-caching for efficiency.
- Improve low-resource translation with backtranslation and multilingual training.

I can (A) add a LaTeX TikZ diagram illustrating the encoder/decoder block-level dataflow, (B) insert full pseudocode for beam search with length normalization and pruning heuristics, or (C) produce a runnable minimal PyTorch example implementing an encoder-decoder transformer for a tiny synthetic MT task.

Which would you like next?