Greedy decoder

In CTC the joint distribution over states factors out into a product of marginals:

$$P(z_{1:T}|\ X_{1:T}, \theta) = \Pi_{t=1}^{T} P(z_t|\ X_{1:T}, \theta)$$

We can take the most likely output at each time-step, which gives us the alignment with the highest probability:

$$\pi^*_{1:T} = \arg\max_{\pi_{1:T}} \Pi_{t=1}^{T} P(z_t = \pi_t|\ X_{1:T})$$

Then merge repeats and remove blanks.

Beam search decoding

Prefix decoding algorithm has 3 nested loops:
- over time - we extend prefixes up to T times
- over prefixes in the beam
- over possible extensions of a prefix

Each prefix can be extended in three possible ways:
- with a blank
- with a repeating character
- with a non-repeating character

We must keep track of two probabilities per prefix:
- The probability of prefix ending with blank $P_{b(t,v)}$.
- The probability of prefix not ending with blank $P_{nb}(t, v)$

Here t denotes time step and v denotes a prefix we got after t time steps.

We start with an empty string prefix:

$$P_b(0, \text{""}) = 1$$

$$P_{nb}(0, \text{""}) = 0$$

If we extend v with a blank, update the probability of ending with a blank:

$$P_{b(t,v)} = P(\varepsilon \mid x_t)...(P_b(t-1, v) + P_{nb}(t-1, v))$$

The prefix v is not updated because blanks are eliminated in the end.

If we extend with a repeat character c, there are two options:
1. The previous symbol is a blank, and now we extend the prefix
2. The previous symbol is not a blank, so we don't extend the prefix (repeats are merged)

In this case, the probability P_{nb} is updated as follows:

$$P_{nb}(t, v + c) = P(c \mid x_t) \cdot P_b(t-1, v)$$

$$P_{nb}(t, v) = P(c \mid x_t) \cdot P_{nb}(t-1, v)$$

Finally, consider extending v at time t with a non-repeat character. It can follow both blank and non-blank characters, so the probability P_{nb} is updated as follows:

$$P_{nb}(t, v + c) = P(c \mid x_t) \cdot (P_b(t-1, v) + P_{nb}(t-1, v))$$

Beam search with LM scores fusion

We may want to apply a language model during decoding, but only when we have a new complete word. This happens when the current symbol is a non-repeat space. As CTC is a discriminative model, LMs can only be integrated as a heuristic:

$$w^* = \arg\max_w \underbrace{P(w \mid X_{1:T})}_{\text{CTC prob}} \cdot \underbrace{P(w)^\alpha}_{\text{LM prob}} \cdot \underbrace{|w|^\beta}_{\text{Length correction}}$$

The formula for an update of P_{nb} when LM is used and the current symbol is a non-repeat space:

$$P_{nb}(t, s + c) = P_{\text{LM}}(s)^\alpha \cdot |s|^\beta \cdot P(c \mid x_t) \cdot (P_b(t-1, s) + P_{nb}(t-1, s))$$

Key differences from standard beam search:
1. Maintain LM scores for each prefix
2. Apply LM scoring when word boundaries are detected
3. Use α to control LM influence and β for word length normalization

Beam search with a second pass LM rescoring

Alternative approach to LM integration:
1. First generate N-best hypotheses using standard beam search
2. Then rescore each hypothesis using LM:

For each hypothesis h in beam:

$$\text{score}(h) = \log P_{\text{CTC}}(h) + \alpha \cdot \log P_{\text{LM}}(h) + \beta \cdot \text{word\_count}(h)$$

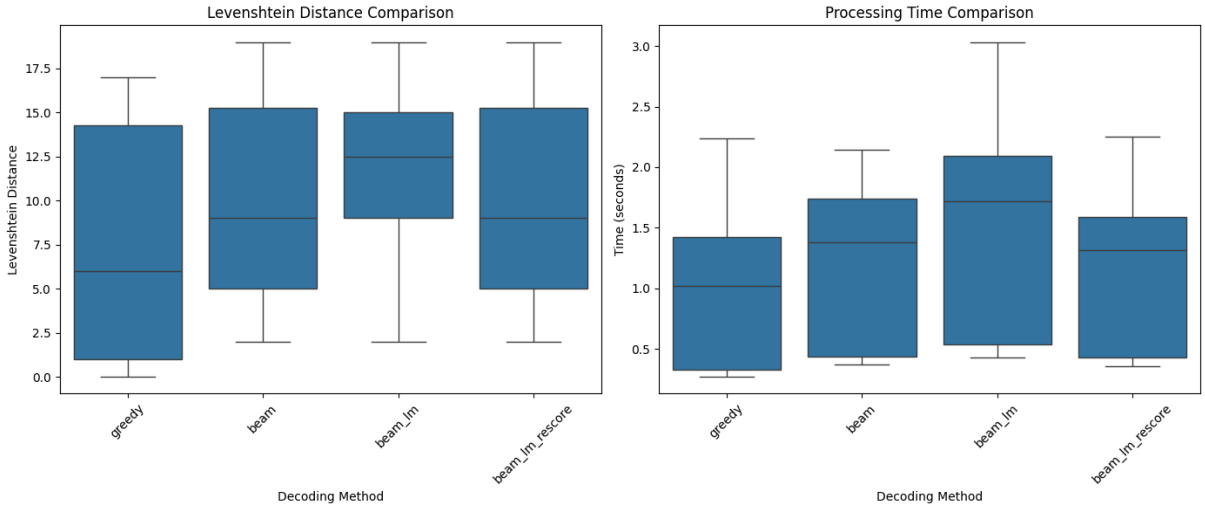
Select hypothesis with maximum score.

Advantages:
- Computationally cheaper than full LM fusion
- Allows using more complex language models
- Easier to tune α and β parameters

Disadvantages:
- Cannot correct prefixes during search
- LM only affects final selection, not search path

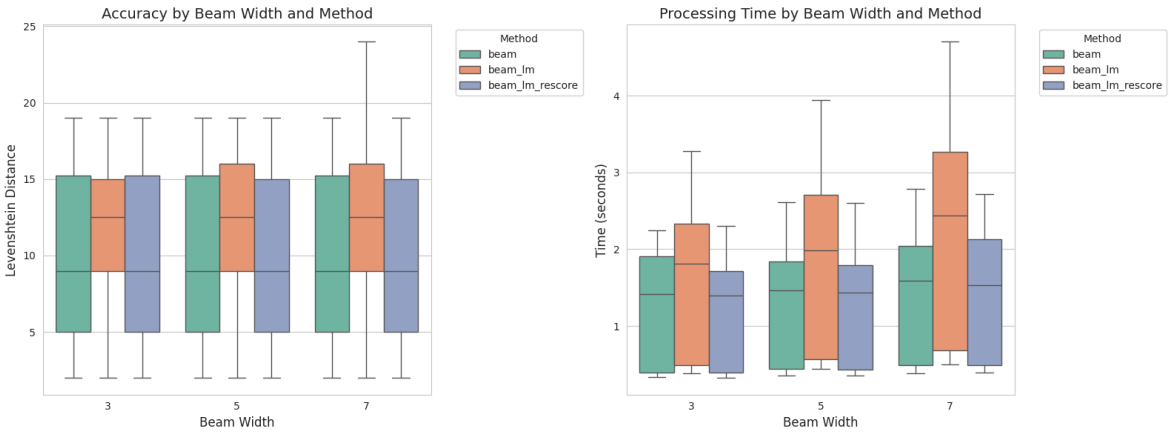
Results

First, just testing the functions with parameters: beam_width=3, alpha=1, beta=1



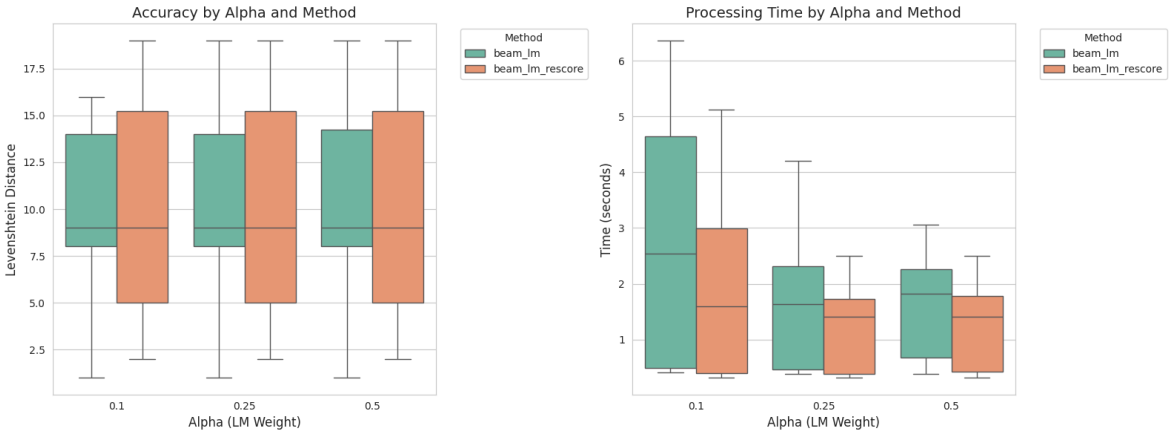
1. Accuracy Performance:
   - Greedy decoding surprisingly delivered the best transcription accuracy
   - Achieved perfect error-free transcription in at least one test case
   - Standard beam search with LM (beam_lm) showed the weakest performance
2. Processing Time:
   - All methods demonstrated comparable average processing times
   - Beam search variants showed slightly longer but generally similar durations
   - Time differences between methods were relatively marginal overall
3. Important Note:
   - The beam_lm's suboptimal results reflect default parameter settings
   - Performance could be significantly improved with proper parameter tuning
   - The method's potential isn't fully represented by these initial results

Second, let's test the model on the different beam_width, exactly: 3, 5 and 7.



As it could be clearly seen, 3 beams are optimal in both terms: time and accuracy.

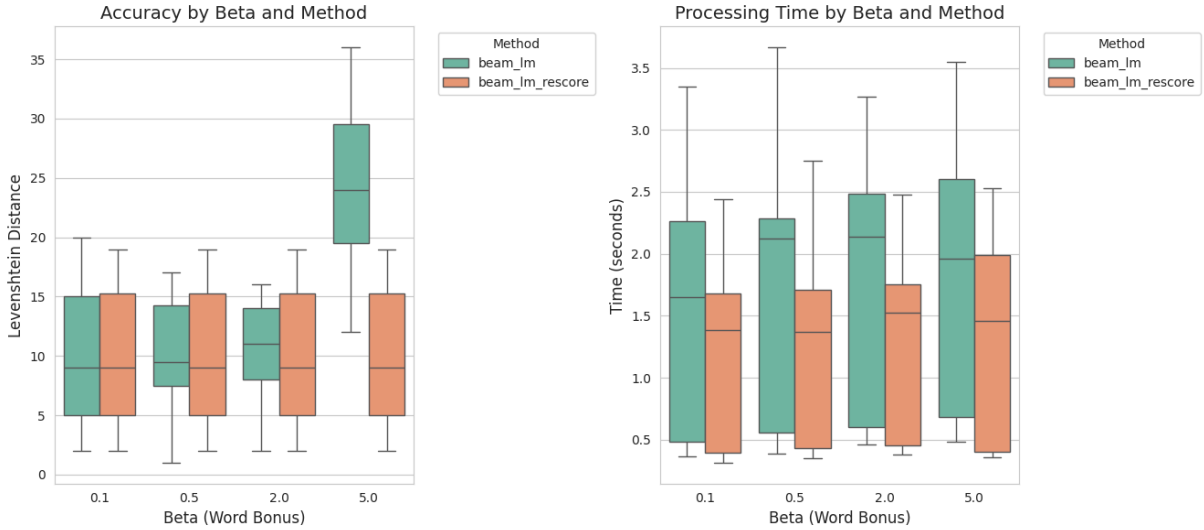
Now, experiments with alpha: 0.1, 0.25, 0,5



- Optimal alpha values: The best Levenshtein distance scores are observed at alpha=0.1

- Performance drop at high alpha: The distance increases significantly, suggesting the language model's strong influence worsens results
- Possible reason: The language model may not be robust enough to handle high alpha values effectively

Final choice: alpha=0.1 was selected for further experiments with beta.



- The beam_lm_rescore method consistently achieves lower Levenshtein distances than beam_lm, indicating better accuracy.
- At a beta value of 5.0, the beam_lm method shows a significant drop in accuracy with high variance.
- The beam_lm_rescore method maintains stable accuracy across all beta values.
- The best accuracy is observed with beam_lm_rescore at beta values of 2.0.
- In terms of processing time, beam_lm_rescore is consistently faster than beam_lm.
- The beam_lm method has higher processing time and greater variability compared to beam_lm_rescore.
- The processing time for both methods does not significantly increase with higher beta values.
- Overall, beam_lm_rescore is the preferred method due to better accuracy and lower processing time.
- The beam_lm method should be avoided at beta 5.0 due to poor performance.