🔹 Q10. Explain the Concept of Perplexity in N-Gram Language Modeling

Why is lower perplexity preferred, and how does it relate to entropy?

✅ What is Perplexity?

Perplexity is a **measurement of uncertainty** in a language model. It tells us **how well the model can predict the next word** in a sequence.

In simple terms:

> **Lower perplexity = better model performance**
> **Higher perplexity = model is "confused" or uncertain**

📌 Mathematical Definition:

Given a language model and a test sentence with N words:

$$\text{Perplexity} = 2^{H(P)} = 2^{-\frac{1}{N}\sum_{i=1}^{N}\log_2 P(w_i|w_{i-n+1}^{i-1})}$$

Where:

- $H(P)$: cross-entropy of the predicted distribution
- $P(w_i \mid w_{i-n+1}^{i-1})$: probability of word w_i given the previous $n-1$ words (N-gram)
- N: total number of words in the test set

🧠 Intuition:

- If a model is certain (e.g., always assigns high probabilities to the correct next word), the perplexity is low.
- If it assigns low probabilities (uncertainty), perplexity is high.

Imagine the model as being "perplexed" — the higher its confusion, the higher the score.

🧪 Example:

Let's say we're testing the model on the sentence:
"The cat sat on the mat"

And the model predicts the next word probabilities like this:

- P("cat" | "The") = 0.5
- P("sat" | "The cat") = 0.4
- P("on" | "cat sat") = 0.6
- ... (and so on)

The perplexity is calculated over all these probabilities. If they're generally **high**, the average log-probability is less negative ⇒ lower perplexity.

🧩 Relation to Entropy:

- **Entropy** measures the average uncertainty in predicting the next word.
- **Perplexity** is simply 2^{Entropy}.

So, **perplexity is the exponentiated form of entropy** — if entropy is high (uncertain predictions), perplexity is also high.

✅ Why Lower Perplexity is Preferred:

- It means the model predicts more confidently and accurately.
- Indicates better generalization on unseen test data.
- Low perplexity implies the model has learned patterns of natural language well.

🔹 Q11. Describe the Steps to Spell Check in SymSpell

✅ What is SymSpell?

SymSpell (Symmetric Delete Spelling Correction) is a **very fast spelling correction algorithm** that uses a **precomputed dictionary of deletions** to suggest correct words for a given misspelling.

It's much faster than traditional edit-distance methods like Levenshtein because it avoids computing edit distances at runtime.

🧰 Core Idea:

Instead of generating all possible candidates **at runtime**, SymSpell precomputes all possible deletions of dictionary words and stores them in a **reverse index**.

🔁 Steps of Spell Checking in SymSpell:

---

1. Dictionary Construction (Preprocessing Phase)

- For each word in the dictionary:
  - Generate all possible words by deleting up to `maxEditDistance` characters.
  - Store these in a hash map where:
    - **Key** = deleted form
    - **Value** = original word(s) that created it

📌 For example, if `"help"` is in the dictionary:

- Generate: `elp`, `hlp`, `hep`, `hel`
- All these forms point back to `"help"`

This step is done once and can be saved.

---

2. Spell Correction (Lookup Phase)

When a misspelled word is entered (e.g., `"hlp"`):

- Generate all possible deletions of the input word.
- Check each form against the precomputed dictionary.
- Return all matching original words as candidates.

---

3. Scoring Candidates

SymSpell scores each candidate using:

- Edit distance between input and candidate word
- Frequency of the candidate word in the corpus (if available)

✅ Best candidates are those **closest in spelling and most frequent**.

---

4. Return the Best Match(es)

- Return top-N best-scoring suggestions.
- You can tune thresholds like:
  - Maximum edit distance
  - Number of suggestions
  - Frequency filters

🧠 Example:

Dictionary: `help`, `held`, `heap`
Misspelled input: `"hlp"`

Generated deletions: `hlp`, `hp`, `lp`, `hp`

- `"hlp"` matches deletion index pointing to `"help"` and `"held"`

Final Suggestions: `"help"`, `"held"`

✅ Benefits of SymSpell:

- Extremely fast: O(1) lookup time with precomputed deletions
- Ideal for large-scale applications (e.g., auto-complete, mobile apps)
- Easy to implement