# Why FastText Outperforms Word2Vec: Handling Rare Words, Subword Information, and OOV Challenges

## word2Vec:

* It treats each word as a unique entity. Words are represented as fixed-size vectors based on their co-occurrence with other words in a given context. However, it does not account for subword information like prefixes or suffixes.
* Pros: Works well for common words, and it captures semantic relationships (like "king" - "man" + "woman" = "queen").
* Cons: Fails to handle out-of-vocabulary (OOV) words and struggles with rare words, typos, or morphologically complex languages.

## FastText:

* It represents each word as a bag of character n-grams, which means it breaks down words into subword units. This allows it to share information between words with similar structures (like “walking” and “walk”).
* Pros: Better at handling OOV words, rare words, and morphologically rich languages. It can generate embeddings for words that were not seen during training by using their subword components.
* Cons: Slightly more computationally intensive due to subword-level processing

### How fastText handle the Out of Vocabulary (OOV) words efficiently as compared to word2Vec?

* Fast text breaks each word into subwords means it break a word into n-grams like walking into walk it use both of these as almost same words with high cosine similarity on the other hand word2Vec make vector every unique word with correspondence to the other almost similar word used in the same corpus. If a word similar to our target word is not in the corpus there is high chance that word2Vec might fails to capture the similarity value or difference between two words.
* Let me show you with the code