P.V.P Siddhartha Institute of Technology (Autonomous)

Department of Computer Science and Engineering

NATURAL LANGUAGE PROCESSING III CSE SEM-II

Descriptive-I Answers Academic Year: 2025-26

Subject Code: 23CS4602C | Regulation: PVP23

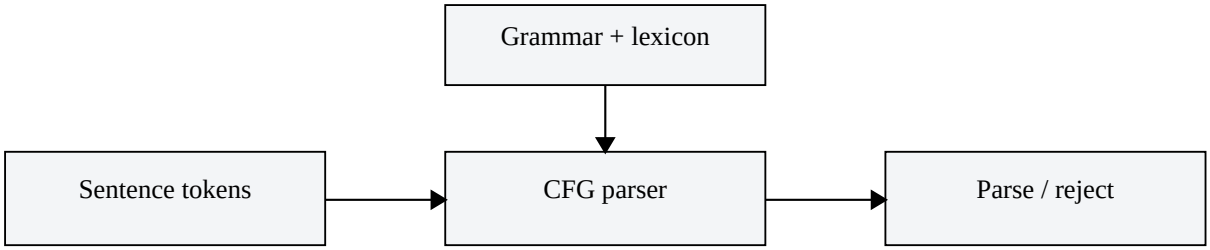
1(a). Build a simple grammar-based language model for a small set of English sentences and explain its working. (CO2-L3) (5M)

Ans:

Definition

A grammar-based language model describes permissible sentence structures using explicit grammatical rules. A simple context-free grammar (CFG) generates or recognizes sentences by rewriting nonterminal symbols until only words remain. A plain CFG specifies grammatical membership; it does not assign probabilities unless extended to a probabilistic grammar.

Architecture / workflow



Step 1: Select a small sentence set

Consider **"The student reads a book", "A teacher reads the book",** and **"The teacher likes a student".** The required word classes are determiners, nouns, and verbs.

Step 2: Construct the grammar

S → NP VP; NP → Det N; VP → V NP

Det → the | a

N → student | teacher | book

V → reads | likes

S is the start symbol; NP and VP represent noun phrase and verb phrase. Det, N, and V represent determiner, noun, and verb. Terminal words are shown in lowercase, with capitalization restored for sentence display.

Step 3: Generate a sentence

S ⇒ NP VP ⇒ Det N VP ⇒ the student VP

⇒ the student V NP ⇒ the student reads NP

⇒ the student reads Det N ⇒ the student reads a book

Step 4: Recognize a sentence

For “A teacher reads the book”, the parser groups “a teacher” as NP and “the book” as NP. It combines “reads” with the object NP to form VP, then combines the subject NP and VP into S. Thus the sentence is accepted by the grammar.

Result and limitations

“Student the reads book a” is rejected because its order does not match the rules. The grammar generalizes beyond the three examples and can also generate “The book reads a teacher”, which is structurally permitted but semantically unusual. Semantic restrictions need additional rules or representations. This illustrates the difference between syntactic well-formedness and meaningfulness.

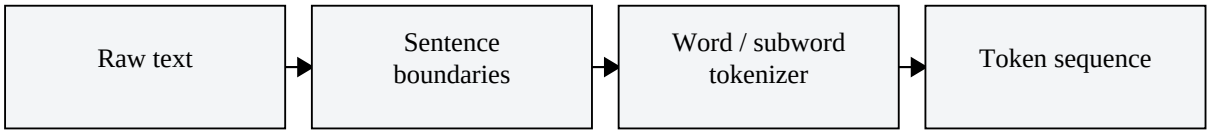
1(b). Apply tokenization and its types to process a given text for natural language processing tasks. (CO2-L3) (5M)

Ans:

Definition

Tokenization divides text into smaller units called tokens, such as sentences, words, punctuation marks, subwords, or characters. The choice of tokens affects vocabulary size and later NLP processing.

Architecture / workflow



Worked text

“Dr. Rao teaches NLP. Students can't miss the 9:30 class!” The examples below use explicit tokenization conventions; different tokenizers may produce different valid outputs.

1. Sentence tokenization

Split the text at sentence boundaries while recognizing that “Dr.” is an abbreviation rather than a complete sentence:
S1: Dr. Rao teaches NLP.
S2: Students can't miss the 9:30 class!

2. Word and punctuation tokenization

A rule-based tokenizer can preserve the abbreviation and time, split punctuation, and divide the contraction into “ca” and “n't”, as in a Penn Treebank-style convention:

[Dr., Rao, teaches, NLP, ., Students, ca, n't, miss, the, 9:30, class, !]

Here “ca” and “n't” are tokenizer outputs, not independent dictionary words. Another tokenizer may preserve “can't” as one token.

3. Whitespace tokenization

Splitting only at spaces yields tokens such as “NLP.” and “class!” with punctuation attached. It is simple but does not properly identify every sentence boundary or contraction.

4. Subword and character tokenization

Subword tokenization: an illustrative split of “unhappiness” is [un, happi, ness]. Algorithms such as BPE or WordPiece learn a vocabulary; actual splits depend on that vocabulary and need not match morpheme boundaries.
Character tokenization: “NLP” becomes [N, L, P], giving a small vocabulary but longer sequences.

Application

Sentence tokens support document segmentation, word tokens support POS tagging and search, and subword tokens help neural models represent rare or new words. Language-specific rules are needed when spaces or punctuation do not reliably mark boundaries.

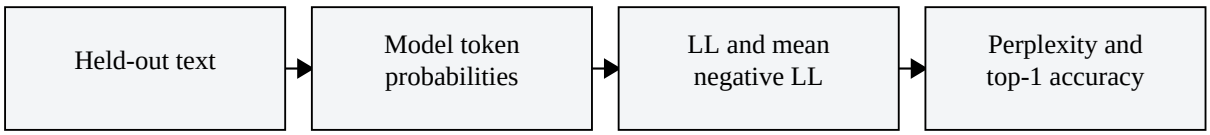
2(a). Apply intrinsic evaluation metrics like log-likelihood, cross-entropy, perplexity, and accuracy for language model evaluation. (CO2-L3) (5M)

Ans:

Definition

Intrinsic evaluation measures a language model directly on held-out text, without embedding it in another application. Log-likelihood, cross-entropy, and perplexity assess the probabilities assigned to the actual words. Next-word accuracy assesses whether the highest-ranked prediction is correct.

Architecture / workflow



Step 1: Define a held-out example

Suppose a model assigns the following conditional probabilities to three scored target tokens. Treat these as illustrative model outputs, excluding any end-of-sentence token from the scored count N = 3.

Observed token	Conditional probability	Highest-ranked prediction
I	0.50	I

Observed token	Conditional probability	Highest-ranked prediction
like	0.25	like
NLP	0.125	ML

Step 2: Compute log-likelihood

$P(w_1,w_2,w_3) = 0.50 \times 0.25 \times 0.125 = 0.015625$

$LL = \Sigma \log_2 P(w_i \mid history_i) = -1 - 2 - 3 = -6$ bits

A higher log-likelihood, meaning less negative on the same test sequence, indicates that the model assigns greater probability to the observed text. Natural logs may also be used, with units then expressed in nats.

Step 3: Compute cross-entropy and perplexity

$H = -(1/N) \times LL = -(-6)/3 = 2$ bits per token

$Perplexity = 2^H = 2^2 = 4$

Lower cross-entropy and perplexity indicate better predictive probability on the same test data and tokenization. Perplexity 4 corresponds to a geometric-mean inverse probability of 4 for the observed tokens.

Step 4: Compute next-word accuracy

Accuracy = correct top-1 predictions / scored tokens

Accuracy = $2/3 \times 100 = 66.67\%$

Accuracy ignores how probability mass is distributed among alternatives; two models can have equal accuracy and different perplexity. Comparisons require consistent tokenization, vocabulary, test text, and sentence-boundary conventions. A zero probability for an observed token makes log-likelihood negative infinity and perplexity infinite unless the model handles unseen events.

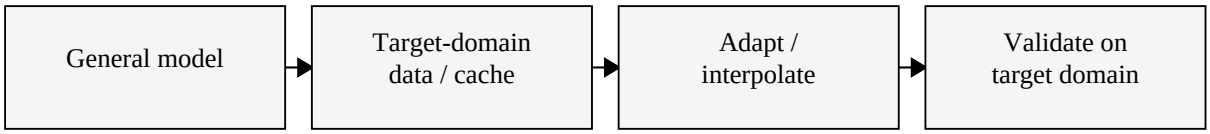
2(b). Utilize language model adaptation methods to handle new vocabulary and contexts. (CO2-L3) (5M)

Ans:

Definition

Language model adaptation adjusts a general model to match a new domain, topic, user, or recent context. It aims to improve predictions when the language in use differs from the original training data.

Architecture / workflow



1. Domain adaptation

A general model may rarely encounter clinical words such as “angioplasty”. Collect representative, permitted domain text and estimate a domain model or continue training a neural model on it. Combine general and domain evidence to avoid losing broad language coverage.

$$P_{adapt}(w \mid h) = \lambda P_{domain}(w \mid h) + (1-\lambda) P_{general}(w \mid h)$$

Suppose $P_{domain}(\text{“angioplasty”} \mid \text{“underwent”}) = 0.40$, $P_{general} = 0.02$, and $\lambda = 0.70$. Then:

$$P_{adapt} = 0.70 \times 0.40 + 0.30 \times 0.02 = 0.286$$

Choose λ using validation data, rather than assuming that a larger domain weight is always better.

2. Topic and user adaptation

Topic adaptation: increase the influence of a sports or technology model when the current document belongs to that topic.
User adaptation: use a consenting user's vocabulary or writing patterns to improve predictions, while limiting overfitting to a small history.

3. Cache-based adaptation

A cache model increases the probability of words appearing in recent context. If “transformer” occurs repeatedly in a lecture, its cache probability rises. Interpolate the cache distribution with the base model so new context affects predictions without discarding the general model.

4. Handling new vocabulary

In a word-based n-gram model, unseen words need a defined treatment: map them to an <UNK> token trained on rare words, or expand the vocabulary and re-estimate probabilities. Smoothing assigns mass to unseen combinations within the defined vocabulary; it does not by itself add entirely new word types. Subword models can represent many new words as sequences of existing pieces.

Example and evaluation

For “The patient underwent ...”, domain adaptation can favor medical continuations. For a software discussion, topic adaptation can favor “deployment” after “production”. Evaluate adaptation on held-out target-domain text using perplexity or an appropriate downstream metric, and check that general performance remains acceptable.

---

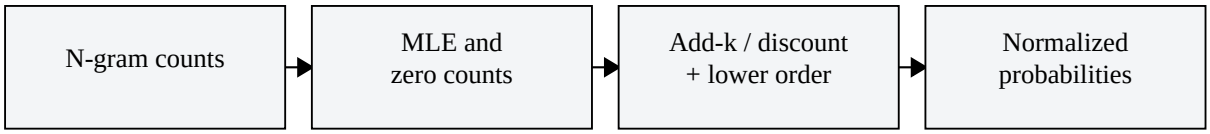
3. Utilize smoothing techniques to handle the zero-frequency problem in n-gram language models. (CO2-L3) (10M)

Ans:

Definition

Smoothing redistributes probability mass from observed events to unseen events so that a valid but unobserved n-gram does not automatically receive probability zero. It is needed because finite training corpora cannot contain every possible word sequence.

Architecture / workflow



Step 1: Identify the zero-frequency problem

$P_{MLE}(w \mid h) = C(h,w) / C(h)$

Consider a toy next-word vocabulary V = {tea, coffee, milk, juice}, with |V| = 4. After the history "drink", suppose the training counts are tea = 3, coffee = 1, milk = 0, and juice = 0. Thus C(drink) = 4.

Next word	Count	MLE probability
tea	3	3/4 = 0.75
coffee	1	1/4 = 0.25
milk	0	0
juice	0	0

The sequence "drink milk" receives zero probability although it is valid. Since sentence probabilities multiply conditional probabilities, one zero makes the entire sentence probability zero.

Step 2: Apply add-one (Laplace) smoothing

$P_{add1}(w \mid h) = (C(h,w)+1) / (C(h)+|V|)$

$\text{Denominator} = 4 + 4 = 8$

Word	Calculation	Smoothed probability
tea	(3+1)/8	0.500
coffee	(1+1)/8	0.250
milk	(0+1)/8	0.125
juice	(0+1)/8	0.125

The four probabilities sum to 1. The model now gives “drink milk” probability 0.125. Add-one is easy to implement but can shift too much mass away from observed n-grams in a large vocabulary.

Step 3: Apply add-k smoothing

$P_{addk}(w \mid h) = (C(h,w)+k) / (C(h)+k|V|)$

For k = 0.5, the denominator is 4 + 0.5 × 4 = 6. The probabilities become tea = 3.5/6 = 0.5833, coffee = 1.5/6 = 0.2500, and milk = juice = 0.5/6 = 0.0833. This makes a smaller adjustment than add-one. Select k on validation data.

Step 4: Apply absolute discounting with interpolation

Subtract a fixed discount D from positive counts and allocate the removed mass through a lower-order distribution. For the same example, set D = 0.5. There are two distinct observed followers of “drink”.

$P_{abs}(w \mid h) = \max(C(h,w)-D,0)/C(h) + \lambda(h)P_{lower}(w)$

$\lambda(h) = D \times N_{+}(h,*) / C(h) = 0.5 \times 2/4 = 0.25$

For arithmetic simplicity, let the lower-order distribution be uniform over the four words, so P_lower(w) = 0.25. The interpolation contribution is 0.25 × 0.25 = 0.0625 for every word.

Word	Discounted term + lower-order term	Probability
tea	2.5/4 + 0.0625	0.6875
coffee	0.5/4 + 0.0625	0.1875
milk	0 + 0.0625	0.0625
juice	0 + 0.0625	0.0625

These probabilities again sum to 1. A real model usually estimates the lower-order distribution from corpus evidence rather than assuming uniformity.

Step 5: Understand Kneser-Ney smoothing

Interpolated Kneser-Ney uses discounting but replaces an ordinary unigram backoff distribution with a **continuation distribution**. A word receives greater lower-order probability when it occurs after many different histories, rather than merely being frequent after one history.

$P_{cont}(w) = N_{+}(*,w) / N_{+}(*,*)$

$N_{+}(*,w)$ counts distinct preceding words for w, and $N_{+}(*,*)$ counts distinct observed bigram types. A word occurring after 10 different predecessors receives continuation probability 10/100 = 0.10 if there are 100 bigram types. This can better model general continuations than raw unigram frequency.

Conclusion

Smoothing solves zero counts within a defined vocabulary. Add-one and add-k add pseudo-counts; discounting and Kneser-Ney combine adjusted higher-order evidence with lower-order estimates. Use held-out perplexity to choose a method and its parameters. Entirely out-of-vocabulary words require additional vocabulary handling.

---

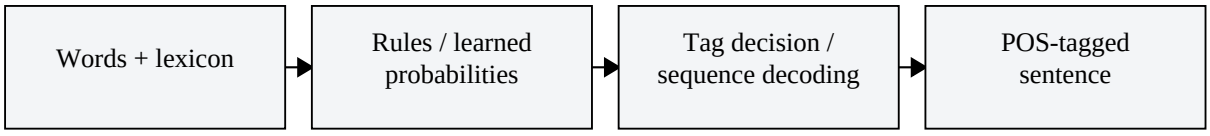
4. Apply rule-based, stochastic, or transformation-based POS tagging to a given sentence. (CO2-L3) (10M)

Ans:

Definition

Part-of-speech (POS) tagging assigns a grammatical category to each word in context. Categories include noun, verb, adjective, adverb, determiner, pronoun, and preposition. A word can receive different tags in different contexts, so tagging requires more than a dictionary lookup.

Architecture / workflow



Step 1: Choose the sentence and tagset

Example: **"The students can book tickets."** Use Penn Treebank-style tags: DT = determiner, NNS = plural noun, MD = modal, VB = base-form verb, and punctuation ".".

Step 2: Apply rule-based tagging

A rule-based tagger combines a lexicon with contextual and morphological rules. First identify possible tags; then apply rules to resolve ambiguous words.

Word	Possible category	Contextual decision
The	Determiner	Tag DT
students	Plural noun	Tag NNS
can	Modal / noun / verb	Modal before the base verb: MD
book	Noun / verb	After a modal, choose VB
tickets	Plural noun / verb	Object of book: NNS
.	Punctuation	Tag .

The/DT students/NNS can/MD book/VB tickets/NNS ./.

The word "book" is a verb here, whereas it is a noun in "The book is useful". A rule such as "choose a base verb reading after a modal" uses surrounding grammar to resolve the ambiguity.

Step 3: Apply stochastic tagging

A stochastic tagger uses probabilities learned from a tagged corpus. In a first-order Hidden Markov Model (HMM), tags are hidden states and words are observed emissions. The tagger chooses the sequence with the largest joint probability of tags and words.

$t^* = \text{argmax}_t \left[\pi(t_1)\, b_{t_1}(w_1) \prod_{i=2}^{n} a_{t_{i-1},t_i}\, b_{t_i}(w_i)\right]$

Here π is the initial-tag distribution, a is the tag-transition probability, and b is the word-emission probability. The Viterbi algorithm finds the best sequence efficiently. Unknown words need an explicit emission strategy, such as suffix features or an unknown-word distribution.

Illustrative local probability comparison

Assume the preceding tag is MD and the ambiguous word is “book”. Suppose P(VB | MD) = 0.80, P(NN | MD) = 0.05, P(book | VB) = 0.02, and P(book | NN) = 0.03.

$$\text{Score}(VB) = 0.80 \times 0.02 = 0.016$$

$$\text{Score}(NN) = 0.05 \times 0.03 = 0.0015$$

This local comparison favors VB. A complete HMM decision considers the entire sentence and all relevant transitions; the local calculation alone is not a full Viterbi decoding.

Step 4: Apply transformation-based tagging

Transformation-based learning, associated with Brill tagging, begins with an initial assignment such as the most frequent tag for each word. It then applies ordered correction rules learned from a tagged training corpus.

Initial output: The/DT students/NNS can/MD book/NN tickets/NNS.
Learned transformation: Change NN to VB when the preceding tag is MD.
Corrected output: The/DT students/NNS can/MD book/VB tickets/NNS.

During training, candidate transformations are evaluated by the number of errors they correct minus the number of new errors they introduce. The best useful rule is applied, and learning continues until the stopping criterion is met. During tagging, the learned rules run in order without access to the test sentence's gold tags.

Comparison and result

Approach	Main evidence	Strength / limitation
Rule-based	Lexicon and manually specified rules	Interpretable; rules need maintenance.
Stochastic	Corpus-estimated probabilities	Handles competing analyses; needs representative data.
Transformation-based	Initial tags plus learned corrections	Readable learned rules; depends on initial tags and training examples.

All three methods can assign the intended sequence for the example. The central decision is that “can” functions as a modal and “book” as a verb. POS tags support parsing, information extraction, lemmatization, and word-sense analysis.

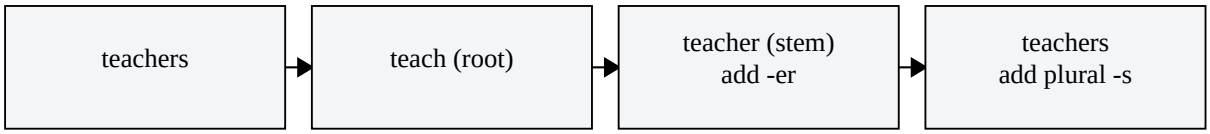
5(a). Construct a morphological structure for given English words by identifying morphemes, roots, stems, and affixes. (CO2-L3) (5M)

Ans:

Definitions

Morpheme: the smallest meaningful or grammatical unit in a word.
Root: the core lexical element that carries the main meaning.
Stem: the form to which an inflectional affix is attached; it can contain a root and derivational affixes.
Affix: a bound morpheme attached to a base, such as a prefix before it or a suffix after it.

Architecture / workflow



Step 1: Segment the words

Word	Morphemes	Root	Structure
unhappiness	un + happy + ness	happy	[[un + happy] + ness]
teachers	teach + er + s	teach	[[teach + er] + s]
replayed	re + play + ed	play	[[re + play] + ed]
books	book + s	book	[book + s]

Step 2: Identify derivation and inflection

unhappiness: happy is a free root; un- is a derivational prefix meaning “not”, and -ness is a derivational suffix forming a noun. The spelling changes from happy to happi before -ness, but the lexical root is happy.
teachers: -er derives the noun “teacher” from the verb “teach”. The plural suffix -s is inflectional. Thus teacher is the stem to which plural -s attaches.
replayed: re- adds the meaning “again”; replay is the stem, and -ed expresses past tense in the intended verbal use.
books: book is both root and stem, and -s marks plural number.

Step 3: Compare root and stem

teachers = ROOT teach + DERIVATION -er + INFLECTION -s

Root = teach; plural stem = teacher; complete word = teachers

The root and stem are identical in “books” but different in “teachers”. Derivation often changes meaning or word class, whereas inflection normally expresses a grammatical form without creating

a new lexeme.

Result

Morphological structure explains how words are built and supports lemmatization, vocabulary analysis, spelling correction, and handling unfamiliar word forms. Not every word can be analyzed correctly by removing an apparent suffix; lexical knowledge is also needed.

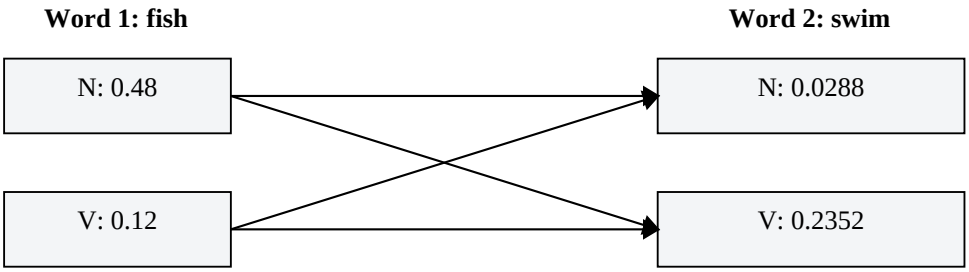
5(b). Make use of the Viterbi algorithm to decode the most likely POS tag sequence in an HMM-based tagger. (CO2-L3) (5M)

Ans:

Definition

Viterbi is a dynamic programming algorithm that finds the highest-probability hidden-state sequence for an observed sequence. In HMM POS tagging, words are observations and POS tags are hidden states.

Architecture / workflow



Best path: N (fish) to V (swim)

Arrows represent transitions; boxes show Viterbi scores.

Step 1: Define a complete toy HMM

Sentence: **"fish swim"**. States are N (noun) and V (verb). Initial probabilities are $\pi(N) = 0.6$ and $\pi(V) = 0.4$. The toy vocabulary contains fish and swim; each emission row sums to 1. No end-state probability is used in this example.

From / To	N	V
N	0.3	0.7
V	0.6	0.4

Emission	fish	swim
N	0.8	0.2
V	0.3	0.7

Step 2: Initialize the first word

$\delta_1(t) = \pi(t) b_t(w_1)$

$\delta_1(N) = 0.6 \times 0.8 = 0.48$; $\delta_1(V) = 0.4 \times 0.3 = 0.12$

Step 3: Recur for "swim" and store backpointers

$\delta_2(t) = \max_s [\delta_1(s) a_{s,t}] \times b_t(swim)$

$\delta_2(N) = \max(0.48 \times 0.3, 0.12 \times 0.6) \times 0.2 = 0.0288$

The maximizing previous tag is N, so psi2(N) = N.

$\delta_2(V) = \max(0.48 \times 0.7, 0.12 \times 0.4) \times 0.7 = 0.2352$

The maximizing previous tag is N, so psi2(V) = N.

Step 4: Terminate and backtrack

At word 2, V has the larger score: 0.2352. Trace its backpointer to N at word 1. The best tagging is **fish/N swim/V**, with joint sequence score 0.2352. This score is not a normalized posterior probability over tag sequences.

Viterbi retains the best path to each state instead of enumerating all paths. For n words and K tags, decoding takes $O(nK^2)$ time; log probabilities are commonly used for numerical stability.

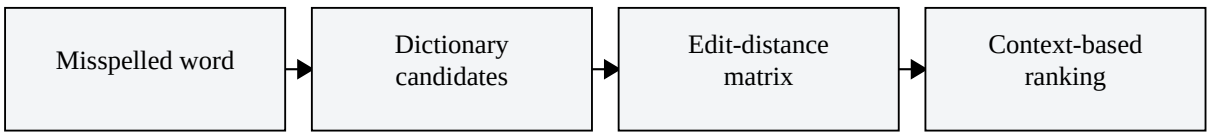
6(a). Solve spelling correction problems by computing edit distance using the Levenshtein distance algorithm. (CO2-L3) (5M)

Ans:

Definition

Levenshtein distance is the minimum number of single-character insertions, deletions, and substitutions required to transform one string into another. Here each operation costs 1 and an exact character match costs 0. Standard Levenshtein distance does not treat transposition as a single operation.

Architecture / workflow



Step 1: Initialize the dynamic programming matrix

Correct the misspelling **"caat"** using candidate **"cat"**. Let D[i,j] be the distance between the first i characters of "caat" and the first j characters of "cat". Initialize D[i,0] = i and D[0,j] = j.

Step 2: Apply the recurrence

D[i,j] = min(D[i-1,j]+1, D[i,j-1]+1, D[i-1,j-1]+cost)

The three terms represent deletion, insertion, and substitution or match. cost = 0 if the characters match, and 1 otherwise.

Step 3: Fill the table

Source / Target	Empty	c	a	t
Empty	0	1	2	3
c	1	0	1	2
a	2	1	0	1
a	3	2	1	1
t	4	3	2	1

For the second “a”, D[3,2] = min(0+1, 2+1, 1+0) = 1. For the last “t”, D[4,3] = min(1+1, 2+1, 1+0) = 1. Therefore the final distance is **D[4,3] = 1**.

Step 4: Recover the edit and choose a correction

One optimal edit is to delete the extra “a”: **caat → cat**. Candidate “coat” is also at distance 1, using substitution of the second character. Edit distance alone therefore cannot uniquely select the intended word.

In **“The caat drank milk”**, contextual evidence favors “cat”. A spelling corrector can generate candidates within a small edit distance and then rank them using word frequency, a language model, and an error model.

Complexity: For strings of lengths m and n, the standard matrix takes O(mn) time and O(mn) space. If only the distance is needed, two rows can reduce memory usage.

6(b). Identify open class and closed class words in the sentence: “The student reads a book in the library.” (CO2-L3) (5M)

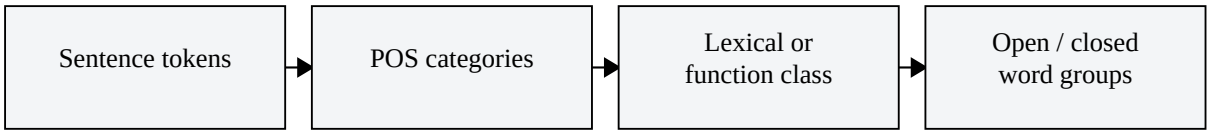
Ans:

Definitions

Open-class words belong to lexical categories that readily admit new members. The main categories are nouns, lexical verbs, adjectives, and adverbs.

Closed-class words belong to categories with relatively limited membership, such as determiners, pronouns, prepositions, conjunctions, and many auxiliary or modal verbs. They mainly express grammatical relationships.

Architecture / workflow



Step 1: Tokenize and assign word classes

Token	POS / category	Class	Function
The	DT / determiner	Closed	Introduces student
student	NN / noun	Open	Names a person
reads	VBZ / lexical verb	Open	Expresses an action
a	DT / determiner	Closed	Introduces book
book	NN / noun	Open	Names an object
in	IN / preposition	Closed	Expresses a location relation
the	DT / determiner	Closed	Introduces library
library	NN / noun	Open	Names a place
.	Punctuation	Neither here	Marks sentence boundary

Step 2: Group and count the words

Open-class word tokens: student, reads, book, library. Total = **4**.
Closed-class word tokens: The, a, in, the. Total = **4**.
The full stop is punctuation and is not counted as a word in either group.

Ignoring case, there are three distinct closed-class word types: the, a, and in. The word “the” occurs twice, so the token count is four even though the type count is three.

Step 3: Explain the distinction with examples

Nouns and verbs carry the main content of this sentence. The vocabulary can easily expand with nouns such as “podcast” or verbs such as “livestream”. Determiners and prepositions organize that content: “the” and “a” specify noun phrases, while “in” connects the event to its location.

Result and application

The sentence contains an equal number of open- and closed-class word tokens. This distinction helps POS tagging, vocabulary analysis, and information retrieval. Closed-class words are not automatically unimportant: prepositions and auxiliaries can be essential for meaning, so deleting all of them is not appropriate for every NLP task.