ASSIGNMENT

Large Language Models (CSE 4357)

Programme: B. Tech (CSE) **Semester: 7th**

Full Marks: 20 **Date of Submission: 20-12-2025**

Subject/Course Learning Outcome	*Taxonomy Level	Question Nos.	Marks
Understand the fundamental concepts of language models, including tokenization and the representation of text as vector embedding for language processing	L2	1	2
Understand and explain the core mechanisms of the Transformer architecture used in modern large language models	L2	2	2
Develop skills to categorize and cluster text data using large language models for text classification and clustering tasks.	L2	3, 4	4
Apply dense retrieval, reranking and retrieval augmented generation methods to enhance traditional keyword-based search systems	L3	5, 6, 7	6
Develop the ability to work with multimodal LLMs by understanding image-to-vector transformations and applying them to visual reasoning tasks	L2	8	2
Understand and implement end-to-end adaptation of LLMs—including data preparation, task-specific fine-tuning, and performance assessment	L2	9, 10	4

*Bloom's taxonomy levels: Knowledge (L1), Comprehension (L2), Application (L3), Analysis (L4), Evaluation (L5), Creation (L6).

Answer all questions. Each question carries equal mark.

- **Assignment scores/markings depend on neatness, clarity and date of submission.**
- **Write your answers with enough detail about your approach and concepts used, so that the grader will be able to understand it easily.**
- **You are allowed to use only those concepts which are covered in the lecture class till date.**

1. Consider a small training corpus with word types and frequencies: low (5), lowest (2), newer (6), and wider (3). Using a SentencePiece tokenization procedure, perform 2 merge iterations and generate the sub-words. At each iteration:
    a. Compute Log- likelihood loss computation.
    b. Find the candidates for removal.
    c. Update the corpus representations and recompute the frequencies for the next iteration.

Ans: Given Corpus,

Word	Frequency
low	5
lowest	2
newer	6
wider	3

Total word tokens = 16

Initial Vocabulary
SentencePiece (Unigram LM) starts with a large vocabulary.
For simplicity, the initial vocabulary includes:
- Whole words: low, lowest, newer, wider
- Characters: l, o, w, e, s, t, n, r, i, d

Iteration 0 (Initial State)
Corpus Representation
low → low
lowest → lowest
newer → newer
wider → wider

Token Frequencies

Token	Frequency
low	5
lowest	2
newer	6
wider	3

Log-Likelihood Computation (Iteration 0)
Let
P(token) = frequency / 16
Log-likelihood:
L_0 = 5 log(5/16) + 2 log(2/16) + 6 log(6/16) + 3 log(3/16)

Iteration 1
Candidate Tokens for Removal
- lowest (frequency = 2)

Reason:
- Lowest frequency
- Can be decomposed into smaller subwords

lowest → low + e + s + t

Token Removed → lowest

Updated Corpus Representation
low → low
lowest → low e s t
newer → newer
wider → wider

Updated Token Frequencies

Token	Frequency
low	7
e	2
s	2
t	2
newer	6
wider	3

Log-Likelihood Computation (Iteration 1)

L_1 = 7 log P(low) + 6 log P(newer) + 3 log P(wider) + 2 [log P(e) + log P(s) + log P(t)]

Iteration 2

Candidate Tokens for Removal

- wider (frequency = 3)

Reason:

- Lower frequency compared to newer
- Can be decomposed into characters

wider → w + i + d + e + r

Token Removed → wider

Updated Corpus Representation

low → low
lowest → low e s t
newer → newer
wider → w i d e r

Updated Token Frequencies

Token	Frequency
low	7
newer	6
w	3
i	3
d	3
e	5
r	3
s	2
t	2

Final Subword Vocabulary After 2 Iterations:

low
newer
w
i
d
e
r
s
t

2. Explain the Transformer architecture, focusing on the self-attention mechanism. Discuss the role of the Query (Q), Key (K), and Value (V) vectors in computing the attention score for a token.

Ans: Transformer Architecture with Emphasis on Self-Attention

The Transformer is a neural network architecture designed for sequence modeling tasks such as machine translation and language modeling. Unlike RNNs or CNNs, the Transformer does not use recurrence or convolution. Instead, it relies entirely on self-attention mechanisms to model dependencies between tokens in a sequence.

The architecture consists of:

- Encoder: Stack of identical layers
- Decoder: Stack of identical layers

Each encoder layer contains:

1. Multi-Head Self-Attention
2. Position-wise Feed-Forward Network

Each decoder layer contains:

1. Masked Multi-Head Self-Attention
2. Encoder–Decoder Attention
3. Position-wise Feed-Forward Network

Residual connections and layer normalization are applied around each sublayer.

Self-Attention Mechanism

Self-attention allows each token in a sequence to attend to all other tokens, including itself, in order to compute a context-aware representation.

For an input sequence of tokens:

x_1 , x_2 , x_3 , …, x_n

each token is projected into three vectors:

- Query (Q)
- Key (K)
- Value (V)

These vectors are learned linear transformations of the input embeddings.

Role of Query (Q), Key (K), and Value (V)

Query (Q)

- Represents what the current token is looking for
- Used to score relevance against all other tokens

Key (K)

- Represents what information each token contains
- Compared with the query to compute similarity

Value (V)

- Contains the actual information to be aggregated
- Weighted and summed to produce the final output

Computation of Attention Scores

Step 1: Linear Projections

For each token embedding

$$Q = xW_Q,\ K = xW_K,\ V = xW_V$$

where $W_Q,\ W_K,\ W_V W_Q,\ W_K,\ W_V$ are learned parameter matrices.

Step 2: Scaled Dot-Product Attention

The attention score between a query and all keys is computed as:

$$Attention(Q,\ K,\ V) = softmax\ (QK^T \diagup \sqrt{d_k})V$$

Where:

- QK^T computes similarity between query and keys
- d_k is the dimensionality of the key vectors
- Scaling by $\sqrt{d_k}$ prevents large dot-product values
- Softmax converts scores into probabilities

Step 3: Weighted Sum of Values

Each value vector is weighted by its corresponding attention score and summed to produce the output

representation for the token.

Interpretation of Self-Attention

- Tokens with higher relevance receive higher attention weights
- Each token dynamically builds a representation based on global context
- Enables modeling of long-range dependencies efficiently
- Parallel computation over all tokens (no sequential dependency)

Multi-Head Attention

Instead of computing a single attention function, the Transformer uses multiple attention heads:

- Each head learns different types of relationships
- Outputs are concatenated and linearly transformed
- Improves expressiveness and representation power

Key Advantages of Self-Attention

- Captures long-distance dependencies effectively
- Fully parallelizable
- Eliminates vanishing gradient issues of RNNs
- Scales well with large datasets

3. Consider three document embeddings:
   D1 = [0.1, 0.5, 0.9]
   D2 = [0.2, 0.6, 0.8]
   D3 = [0.9, 0.2, 0.4]
   a. Calculate pairwise Cosine similarity, Euclidean distance, and Dot product between all documents.
   b. Explain which similarity metric is generally preferred for clustering text embeddings in high-dimensional spaces.

Ans: Given Document Embeddings

D1 = [0.1, 0.5, 0.9]
D2 = [0.2, 0.6, 0.8]
D3 = [0.9, 0.2, 0.4]

(a) Pairwise Similarity Calculations

Step 1: Vector Norms

$\|D1\| = \sqrt{(0.1^2 + 0.5^2 + 0.9^2)}$
$\|D1\| = \sqrt{(0.01 + 0.25 + 0.81)} = \sqrt{1.07} \approx 1.034$
$\|D2\| = \sqrt{(0.2^2 + 0.6^2 + 0.8^2)}$
$\|D2\| = \sqrt{(0.04 + 0.36 + 0.64)} = \sqrt{1.04} \approx 1.020$
$\|D3\| = \sqrt{(0.9^2 + 0.2^2 + 0.4^2)}$
$\|D3\| = \sqrt{(0.81 + 0.04 + 0.16)} = \sqrt{1.01} \approx 1.005$

1. Dot Product

Pair	Dot Product
D1 · D2	(0.1×0.2) + (0.5×0.6) + (0.9×0.8) = 1.04
D1 · D3	(0.1×0.9) + (0.5×0.2) + (0.9×0.4) = 0.55
D2 · D3	(0.2×0.9) + (0.6×0.2) + (0.8×0.4) = 0.62

2. Cosine Similarity

Formula:
$\cos(\theta) = (A \cdot B) / (\|A\| \|B\|)$

Pair	Cosine Similarity
D1, D2	$1.04 / (1.034 \times 1.020) \approx$ **0.986**
D1, D3	$0.55 / (1.034 \times 1.005) \approx$ **0.529**
D2, D3	$0.62 / (1.020 \times 1.005) \approx$ **0.605**

3. Euclidean Distance

Formula:
$d(A,B) = \sqrt{\Sigma(A_i - B_i)^2}$

Pair	Euclidean Distance
D1, D2	$\sqrt{[(-0.1)^2 + (-0.1)^2 + (0.1)^2]} = \sqrt{0.03} \approx 0.173$
D1, D3	$\sqrt{[(-0.8)^2 + (0.3)^2 + (0.5)^2]} = \sqrt{0.98} \approx 0.990$
D2, D3	$\sqrt{[(-0.7)^2 + (0.4)^2 + (0.4)^2]} = \sqrt{0.81} = 0.900$

Summary Table

Pair	Dot Product	Cosine Similarity	Euclidean Distance
D1 – D2	1.04	0.986	0.173
D1 – D3	0.55	0.529	0.990
D2 – D3	0.62	0.605	0.900

(b) Preferred Similarity Metric for Clustering Text Embeddings

Cosine similarity is generally preferred for clustering text embeddings in high-dimensional spaces.\

Reasons:

1. Magnitude independence: Text embeddings often vary in length; cosine similarity focuses on

direction, not vector size.
2. Semantic relevance: Embedding models encode meaning in vector direction rather than absolute values.
3. Curse of dimensionality: In high dimensions, Euclidean distances become less informative, while cosine similarity remains discriminative.
4. Empirical standard: Most NLP clustering methods (e.g., document similarity, sentence embeddings) use cosine similarity by default.

4. Explain how prompting affects the performance of large language models and provides examples of different types of prompting.

Ans: Effect of Prompting on Large Language Models
Prompting is the way input instructions are given to a large language model (LLM). Since LLMs generate outputs based entirely on the provided prompt, **the quality, clarity, and structure of the prompt directly affect model performance**. Well-designed prompts improve task understanding, reasoning accuracy, output format, and relevance, while vague prompts lead to incorrect or unfocused responses.

Prompting helps in:
- Clearly defining the task
- Controlling output structure and length
- Improving logical reasoning
- Reducing ambiguity and errors

Types of Prompting

1. Zero-Shot Prompting: Task given without examples.
Example: Translate the sentence into French.

2. **One-Shot Prompting:** One example is provided before the task.
Example: Input: Good → Positive; Input: Bad → ?

3. Few-Shot Prompting: Multiple examples guide the model.
Example: Several labeled sentiment examples before classification.

4. Chain-of-Thought Prompting: Model is asked to explain reasoning step by step.
Example: Solve the problem step by step.

5. Instruction-Based Prompting: Direct command specifying the task.
Example: Summarize the paragraph in 3 points.

5. Compare semantic search with keyword-based search in terms of scalability, accuracy, and computational cost. Provide one example where keyword search may still be preferable. What is the role of embeddings in semantic search pipelines? How do they capture meaning beyond keywords?

Ans: Semantic Search vs Keyword-Based Search

Aspect	Keyword-Based Search	Semantic Search
Accuracy	Matches exact words or patterns; fails on synonyms and paraphrases	Understands intent and context; handles synonyms and rewording
Scalability	Highly scalable using inverted indexes; works well on very large corpora	Scales with vector indexes (ANN), but requires embedding storage and indexing
Computational Cost	Low at query time; simple string matching	Higher cost due to embedding generation and vector similarity computation

Example Where Keyword Search is Preferable

Legal or compliance search

When exact terms must match (e.g., searching for a specific clause number, regulation ID, error code, or API function name), keyword search is more reliable than semantic search.

Role of Embeddings in Semantic Search

Embeddings are dense numerical vector representations of text generated by neural models. They map words, sentences, or documents into a continuous vector space where semantically similar texts are close to each other, even if they share no exact keywords.

How Embeddings Capture Meaning Beyond Keywords

- Encode context, not just word presence
- Capture synonyms and paraphrases (e.g., “car” ≈ “automobile”)
- Preserve semantic relationships and intent
- Reduce language variability into geometric proximity in vector space

Example:

“cheap flights” and “low-cost airfare” have different keywords but similar embeddings.

6. How does the perplexity metric help in evaluating the quality of text generated by a language model? Explain briefly. Additionally, a language model assigns probabilities of 0.25, 0.10, 0.20, and 0.05 to a four-token sequence. Using these probabilities, calculate the perplexity of the model for this sequence and show the formula and final numerical value.

Ans: Perplexity measures how well a language model predicts a sequence of tokens.
It is the inverse of the geometric mean of the predicted probabilities.

- Lower perplexity → better model
- Indicates less uncertainty in predicting the next token
- Commonly used to compare language models on the same dataset

Perplexity does not directly measure fluency or factual correctness, but it is a strong indicator of predictive quality.

Perplexity Formula

For a sequence of *N* tokens with probabilities $p_1, p_2, \ldots, p_N$:

$$\text{Perplexity} = \left(\prod_{i=1}^{N} p_i \right)^{-\frac{1}{N}}$$

Given Probabilities

$p_1 = 0.25$
$p_2 = 0.10$
$p_3 = 0.20$
$p_4 = 0.05$
$N = 4$

Calculation

Product of probabilities: $0.25 \times 0.10 \times 0.20 \times 0.05 = 0.00025$

Perplexity:

$PPL = (0.00025)^{-1/4}$

≈ 7.95

Perplexity ≈ 8

7. The figure below shows the output of an information retrieval system on two queries. Crosses correspond to the relevant documents, dashes to non- relevant documents. Let the two documents contain 3 and 6 relevant documents, respectively, but only those shown in the figure are retrieved by the system, not the others.

Rank	1	2	3	4	5	6	7	8	9	10
Q1	X			X	X					
Q2		X		X		X	X		X	

a. Calculate precision at K (where K=5)
b. Compute the average precision(AP)
c. Compute the mean average precision(MAP)
d. Normalized Discounted Cumulative Gain (nDCG) (where relevance grade is binary)

Ans: Given
- Two queries: Q1 and Q2
- Relevant documents:
  - Q1 has 3 relevant documents in total
  - Q2 has 6 relevant documents in total
- Only the documents shown in the ranking are retrieved
- Relevance is binary (X = relevant, blank = non-relevant)

Retrieved Rankings
Q1

Rank	1	2	3	4	5
Relevance	X	–	–	X	X

Relevant retrieved at ranks 1, 4, 5

Q2

Rank	1	2	3	4	5	6	7	8	9	10
Relevance	–	X	–	X	–	X	X	–	X	–

Relevant retrieved at ranks 2, 4, 6, 7, 9

(a) Precision at K (K = 5)
Formula

$$P@5 = \text{Relevant documents in top } 5 / 5$$

Q1
Relevant in top 5 = 3

$$P@5(Q1) = 3/5 = 0.6$$

Q2
Relevant in top 5 = 2 (ranks 2 and 4)

$$P@5(Q2) = 2/5 = 0.4$$

(b) Average Precision (AP)
Formula

$$AP = (1/\text{Total relevant documents}) \sum P@k$$

AP for Q1 (Total relevant = 3)

Rank	Relevant	Precision
1	Yes	1/1 = 1.00
4	Yes	2/4 = 0.50
5	Yes	3/5 = 0.60

$$AP(Q1) = (1.00+0.50+0.60) / 3 = 2.10/3 = 0.70$$

AP for Q2 (Total relevant = 6)

Rank	Relevant	Precision
2	Yes	1/2 = 0.50
4	Yes	2/4 = 0.50
6	Yes	3/6 = 0.50
7	Yes	4/7 ≈ 0.571
9	Yes	5/9 ≈ 0.556

$$AP(Q2) = \frac{0.50 + 0.50 + 0.50 + 0.571 + 0.556}{6}$$

$$AP(Q2) \approx \frac{2.627}{6} \approx 0.438$$

(c) Mean Average Precision (MAP)
Formula

$$MAP = \frac{AP(Q1) + AP(Q2)}{2}$$

$$MAP = \frac{0.70 + 0.438}{2} \approx 0.569$$

(d) Normalized Discounted Cumulative Gain (nDCG)
DCG Formula (binary relevance)

$$DCG = \sum_{i=1}^{N} \frac{rel_i}{\log_2(i+1)}$$

nDCG for Q1
Relevant at ranks 1, 4, 5

$$DCG(Q1) = 1 + \frac{1}{\log_2 5} + \frac{1}{\log_2 6}$$

$$DCG(Q1) \approx 1 + 0.431 + 0.387 = 1.818$$

Ideal DCG (IDCG): all 3 relevant docs at top ranks (1,2,3)

$$IDCG(Q1) = 1 + \frac{1}{\log_2 3} + \frac{1}{\log_2 4}$$

$$IDCG(Q1) \approx 1 + 0.631 + 0.5 = 2.131$$

$$nDCG(Q1) = \frac{1.818}{2.131} \approx 0.853$$

nDCG for Q2
Relevant at ranks 2, 4, 6, 7, 9

$$DCG(Q2) = \frac{1}{\log_2 3} + \frac{1}{\log_2 5} + \frac{1}{\log_2 7} + \frac{1}{\log_2 8} + \frac{1}{\log_2 10}$$

$$DCG(Q2) \approx 0.631 + 0.431 + 0.356 + 0.333 + 0.301 = 2.052$$

IDCG(Q2): 6 relevant docs ideally at ranks 1–6

$$IDCG(Q2) \approx 1 + 0.631 + 0.5 + 0.431 + 0.387 + 0.356 = 3.305$$

$$nDCG(Q2) = \frac{2.052}{3.305} \approx 0.621$$

Final Answers Summary

Metric	Q1	Q2
Precision@5	0.6	0.4
Average Precision (AP)	0.70	0.438
nDCG	0.853	0.621

MAP ≈ 0.569

8. Explain what makes a language model “multimodal.” How does it differ from unimodal text-only LLMs?

Ans: A multimodal language model is a model that can process, understand, and generate information across multiple data modalities, such as text, images, audio, video, or structured data, within a single unified system.

What Makes a Model Multimodal

A language model is considered multimodal if it:

- Accepts non-text inputs (e.g., images, speech)
- Combines information from different modalities
- Produces outputs that depend on cross-modal understanding

Example:

Answering a question about an image or describing a scene shown in a picture.

Difference from Unimodal (Text-Only) LLMs

Aspect	Unimodal LLM	Multimodal LLM
Input	Text only	Text + image/audio/video
Understanding	Linguistic patterns only	Cross-modal semantic understanding
Output	Text	Text (conditioned on multiple modalities)
Capability	Cannot interpret images or audio	Can reason over visual or auditory content

9. State and compare sentence-level embeddings with document-level embeddings. Which would be more suitable for a legal search engine, and why?

Ans: Sentence-Level Embeddings: Sentence-level embeddings represent the meaning of individual sentences as dense vectors. They capture fine-grained semantics and are useful when precise phrasing or specific statements matter.
Characteristics:

- Short text units (sentences)
- High semantic precision
- Sensitive to wording and context
- Useful for question answering and clause matching

Document-Level Embeddings: Document-level embeddings represent the overall meaning of an entire document by aggregating information across multiple sentences or paragraphs.
Characteristics:

- Long text units (documents)
- Coarse, holistic representation
- Less sensitive to individual sentence details
- Useful for topic-level retrieval and clustering

Comparison

Aspect	Sentence-Level	Document-Level
Granularity	Fine-grained	Coarse
Text length	Short	Long
Precision	High	Lower
Context scope	Local	Global
Typical use	QA, clause search	Topic search, clustering

Suitable Choice for a Legal Search Engine
Sentence-level embeddings are more suitable.
Reason: Legal search requires exact clauses, obligations, definitions, and precedents, not just topic similarity. Sentence-level embeddings allow precise retrieval of relevant legal statements, while document-level embeddings may miss critical details by averaging meanings across the document.

10. Explain the difference between fine-tuning, instruction-tuning, and RLHF.
How do these methods improve LLM performance and safety?

Ans: Fine-Tuning: Fine-tuning adapts a pre-trained language model to a specific task or domain by further training it on labeled data.
Purpose:
- Improve task accuracy
- Inject domain-specific knowledge (e.g., legal, medical)

Example:
Training a general LLM on legal documents for legal text analysis.

Instruction-Tuning: Instruction-tuning trains the model on instruction–response pairs so it learns to follow human-written commands.
Purpose:
- Improve instruction-following ability
- Make outputs more useful and structured

Example:
Training on prompts like "Summarize the text" → correct summary.

Reinforcement Learning from Human Feedback (RLHF): RLHF uses human preference judgments to guide the model's behavior via reinforcement learning.
Purpose:
- Align outputs with human values
- Reduce harmful, biased, or unsafe responses

Example:
Humans rank model responses; preferred responses receive higher rewards.

Comparison

Aspect	Fine-Tuning	Instruction-Tuning	RLHF
Training Signal	Labeled data	Instructions + responses	Human preferences
Goal	Task/domain accuracy	Obedience to instructions	Alignment & safety
Output Style	Task-specific	Helpful & structured	Safer & aligned
Safety Impact	Low	Moderate	High

How They Improve Performance and Safety
- Fine-tuning improves factual accuracy in specific domains
- Instruction-tuning improves usability and controllability
- RLHF improves safety, reduces harmful outputs, and aligns responses with human expectations