R V College of Engineering
Department of Computer Science and Engineering
CIE 1: Scheme

Subject : (Code)	**Natural Language Processing (Group B) (IS355TBC)**	**Semester: 5th BE**	
Date :	**Duration: 120** minutes	**Staff: Dr.Sindhu D V**	
Name :	**USN** :	**Section :**	**A**

512

Q. No	**Part-A**	**Marks**	**Level**	**CO**

1	def tweet_length(tweet):
	    return len(tweet)
	# Example usage:
	tweet = "Learning Python is fun! #coding #python"
	length = tweet_length(tweet)
	print(f"The length of the tweet is: {length} characters.")
		2	2	2

2	from transformers import AutoTokenizer
	model_name = "bert-base-uncased" # Replace with your desired model
	tokenizer = AutoTokenizer.from_pretrained(model_name)
	text = "Transformers are powerful tools in NLP."
	tokens = tokenizer(text)
		2	1	1

3	**S → VP PP** **VP → V** **PP → PREP NP** **NP → ADJ NP | N | N NUM** **V → 'welcome'** **PREP → 'to'** **ADJ → 'prosperous' | 'new'** **N → 'year'** **NUM → '2025'**	2	3	2
4	To enable the model to focus on different parts of the input sentence when processing each word	1	1	1
5	Distributed Training Large models such as GPT and BERT require vast computational resources, so they are typically trained using distributed training across multiple GPUs or machines to speed up the process.	2	3	1

Q. No	Answer all questions	Marks	Level	CO
1	Consider the sentences like “The dog chased the cat" "A cat saw a dog" CFG S → NP VP NP → Det N | N VP → V NP | V Det → 'the' | 'a' N → 'dog' | 'cat' V → 'chased' | 'saw'	10	3	3

```
class RecursiveParser:
    def __init__(self, tokens):
        self.tokens = tokens
        self.position = 0
        self.parse_tree = []
    def parse_S(self):
        # S → NP VP
        start_pos = self.position
        if self.parse_NP() and self.parse_VP():
            return True
        self.position = start_pos
        return False

    def parse_NP(self):
        # NP → Det N | N
        start_pos = self.position
        if self.parse_Det() and self.parse_N():
            return True
        self.position = start_pos
        if self.parse_N():
            return True
        self.position = start_pos
        return False

    def parse_VP(self):
        # VP → V NP | V
        start_pos = self.position
```

```
        if self.parse_V() and self.parse_NP():
            return True
        self.position = start_pos
        if self.parse_V():
            return True
        self.position = start_pos
        return False

    def parse_Det(self):
        # Det → 'the' | 'a'
        if self.match('the') or self.match('a'):
            self.parse_tree.append(('Det', self.tokens[self.position - 1]))
            return True
        return False

    def parse_N(self):
        # N → 'dog' | 'cat'
        if self.match('dog') or self.match('cat'):
            self.parse_tree.append(('N', self.tokens[self.position - 1]))
            return True
        return False

    def parse_V(self):
        # V → 'chased' | 'saw'
        if self.match('chased') or self.match('saw'):
            self.parse_tree.append(('V', self.tokens[self.position - 1]))
            return True
        return False

    def match(self, token):
        # Match a token
        if self.position < len(self.tokens) and self.tokens[self.position] == token:
            self.position += 1
            return True
        return False

# Example usage:
sentence = "the dog chased the cat"
tokens = sentence.split()
parser = RecursiveParser(tokens)
```

<table>
<tr>
<td></td>
<td>

```
if parser.parse_S() and parser.position == len(tokens):
    print("Valid sentence!")
    print("Parse Tree:", parser.parse_tree)
else:
    print("Invalid sentence.")
```

```
          S
        /   \
      NP     VP
    /  \    /  \
  Det   N  V    NP
  |     |  |   /  \
 the   dog chased Det  N
                  |    |
                 the  cat
```

</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>

Word Sense Disambiguation (WSD)
WSD is the task of determining the intended meaning of a word in a given context. This involves:

- **Lexical Resources:** Dictionaries or WordNet for listing possible senses.
- **Contextual Analysis:** Utilizing surrounding words and sentence structure to infer the correct sense.

Machine Translation (MT)
MT systems convert text between languages. Without WSD, translations often fail when faced with polysemous words, leading to incorrect or awkward translations. Incorporating WSD helps address these issues, especially in neural models like Transformer-based architectures.

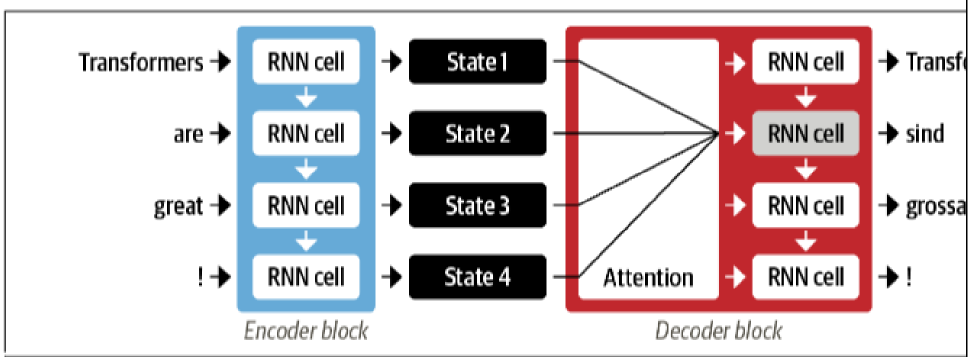
Example 1: "Bank"

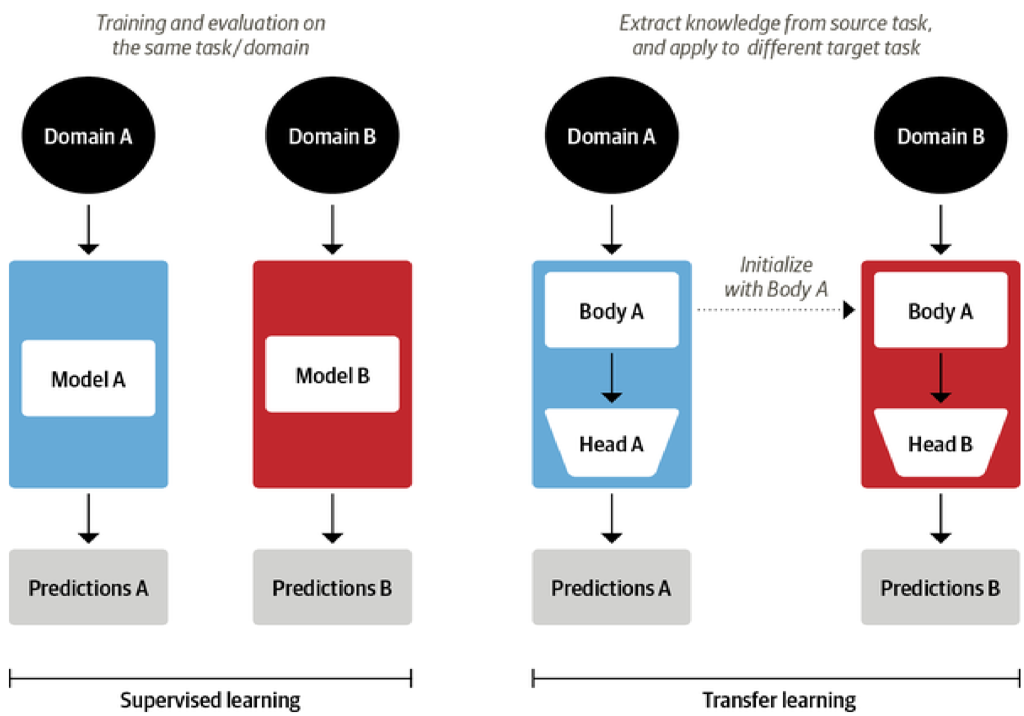
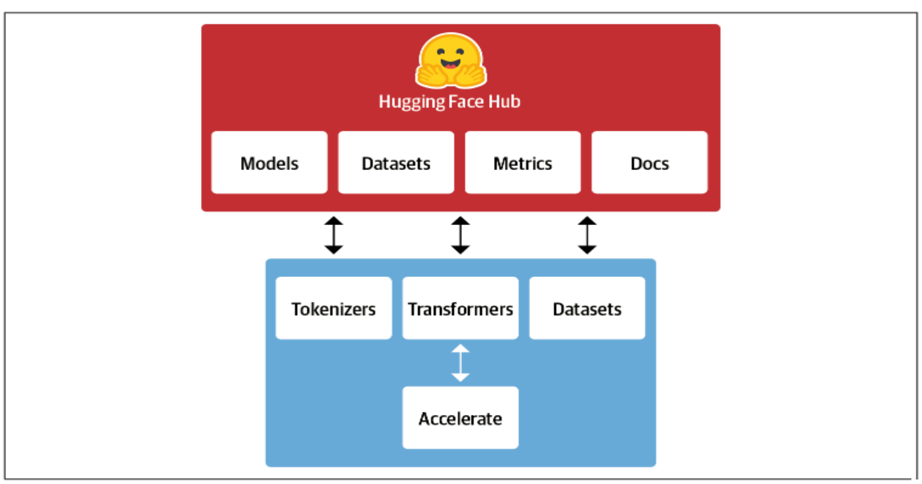
- **Source Sentence:** "He sat by the bank to watch the sunset."
- **Baseline MT Output (French):** "Il s'est assis près de la banque pour regarder le coucher du soleil." (*Incorrect; "banque" refers to a financial institution*)
- **WSD-Augmented MT Output (French):** "Il s'est assis près de la rive pour regarder le coucher du soleil." (*Correct; "rive" refers to the riverbank*)

Example 2: "Charge"

- **Source Sentence:** "The company will charge you for extra services."
- **Baseline MT Output (Spanish):** "La empresa te cargará por servicios extra." (*Incorrect; "cargar" suggests a physical burden*)
- **WSD-Augmented MT Output (Spanish):** "La empresa te cobrará por

</td>
<td>10</td>
<td>4</td>
<td>2</td>
</tr>
</table>

	servicios extra." (*Correct; "cobrar" refers to monetary charges*)			
3	The main idea behind attention is that instead of producing a single hidden state for the input sequence, the encoder outputs a hidden state at each step that the decoder can access. However, using all the states at the same time would create a huge input for the decoder, so some mechanism is needed to prioritize which states to use. This is where attention comes in: it lets the decoder assign a different amount of weight, or "attention," to each of the encoder states at every decoding timestep. This process is illustrated in Figure 1-4, where the role of attention is shown for predicting the third token in the output sequence.  *Figure 1-4. An encoder-decoder architecture with an attention mechanism for RNNs* By focusing on which input tokens are most relevant at each timestep, these attention-based models are able to learn nontrivial alignments between the words in a generated translation and those in a source sentence. Although attention enabled the production of much better translations, there was still a major shortcoming with using recurrent models for the encoder and decoder: the computations are inherently sequential and cannot be parallelized across the input sequence. With the transformer, a new modelling paradigm was introduced: dispense with recurrence altogether, and instead rely entirely on a special form of attention called self-attention. The basic idea is to allow attention to operate on all the states in the same layer of the neural network.	2m×5=10	2	1
4	It is nowadays common practice in computer vision to use transfer learning to train a convolutional neural network like ResNet on one task, and then adapt it to or fine-tune it on a new task. This allows the network to make use of the knowledge learned from the original task. Architecturally, this involves	6	1,2	1

	splitting the model into of a body and a head, where the head is a task-specific network. During training, the weights of the body learn broad features of the source domain, and these weights are used to initialize a new model for the new task. Compared to traditional supervised learning, this approach typically produces high-quality models that can be trained much more efficiently on a variety of downstream tasks, and with much less labeled data. A comparison of the two approaches 			
5	 Challenges 1.Language NLP research is dominated by the English language. There are several models for other languages, but it is harder to find pretrained models for rare or low-	10	3	2

	resource languages. 2. Data availability Although we can use transfer learning to dramatically reduce the amount of labeled training data our models need ; it is still a lot compared to how much a human needs to perform the task. 3. Working with long documents Self-attention works extremely well on paragraph-long texts, but it becomes very expensive when we move to longer texts like whole documents. 4. Opacity As with other deep learning models, transformers are to a large extent opaque. It is hard or impossible to unravel “why” a model made a certain prediction. This is an especially hard challenge when these models are deployed to make critical decisions. 5. Bias Transformer models are predominantly pretrained on text data from the internet. This imprints all the biases that are present in the data into the models.			

Marks Distribution	Particulars		CO1	CO2	CO3	CO4	L1	L2	L3	L4	L5	L6
	Test	Max Marks	28	19	13		6	26	18	10		

BT-Blooms Taxonomy, CO-Course Outcomes, M-Marks