# Natural Language Processing
## UNIT I

Adarsh Kumar
Department of Industrial Mining Engineering and ICT (EMIT),
Manresa School of Engineering (EPSEM),
Polytechnic University of Catalonia, Manresa, Barcelona, Spain
kumar.adarsh@upc.edu

May 5, 2026

# Motivation

- Natural Language Processing (NLP) enables computers to process, understand, and generate human language in text and speech.
- Applications: machine translation, search, chatbots, sentiment analysis, summarization, and code assistants.
- Recent progress is driven by deep learning, transformers, and large language models (LLMs).

# Course Focus

- **Foundations of NLP**: language, text processing, classical models.
- **Neural approaches**: embeddings, sequence models, core tasks.
- **Transformers and LLMs**: architecture, prompt engineering, and safety.

# What is NLP?

- Interdisciplinary field at the intersection of computer science, AI, and linguistics.
- Goal: build systems that can analyze and generate language in a way that is useful and reliable.
- Tasks range from low-level tokenization to high-level reasoning and dialogue.

# Language and Linguistic Levels

- Morphology: structure of words (prefixes, stems, suffixes).
- Syntax: how words combine into grammatical sentences.
- Semantics and pragmatics: meaning, context, and intent in language.

# Text Processing Pipeline

- Basic steps: normalization, tokenization, sentence splitting.
- Handling vocabulary: stopwords, punctuation, casing, lemmas and stems.
- Subword methods (BPE, WordPiece) to handle rare words and open vocabulary.

# Classical NLP Models

- Bag-of-words and TF–IDF representations for documents.
- n-gram language models with smoothing (e.g., bigram, trigram models).
- Classical classifiers: Naive Bayes, logistic regression, SVMs for text classification.

# Overview

- Core Python NLP libraries.
- Deep learning and transformer toolkits.
- Data handling and experiment tracking.
- LLM and prompt-engineering tools.

# Core Python NLP Libraries

- **NLTK**: classic toolkit for tokenization, tagging, parsing, and educational NLP demos.
- **spaCy**: fast, industrial-strength NLP (tokenization, POS, NER, dependency parsing).
- **Gensim**: topic modelling (LDA) and document similarity with large corpora.
- **TextBlob**: simple sentiment analysis and quick prototypes.
- **scikit-learn**: TF–IDF, feature extraction, and classical ML models for text.

# Why These Libraries?

- Cover the full classical pipeline: preprocessing, features, and supervised learning.
- Widely used in tutorials, MOOCs, and textbooks for introductory NLP.
- Integrate easily with Jupyter and data-science workflows.

# Deep Learning & Transformers

- **Hugging Face Transformers**: pretrained models (BERT, RoBERTa, GPT, T5, etc.) and task-specific pipelines.
- **Sentence-Transformers**: sentence and document embeddings for semantic search and clustering.
- **PyTorch** / **TensorFlow/Keras**: frameworks to build and fine-tune neural NLP models.
- **AllenNLP**: research-friendly library on top of PyTorch for advanced NLP experiments.

# Typical Deep NLP Workflows

- Fine-tune a pretrained transformer on text classification or NER.
- Build semantic similarity or retrieval systems using sentence embeddings.
- Prototype and compare custom architectures in PyTorch or Keras.

# Data Handling & Evaluation

- **Pandas** & **NumPy**: data cleaning, manipulation, and feature analysis.
- **Matplotlib** / **Seaborn**: visualization of text statistics and model metrics.
- **Hugging Face Datasets**: ready-to-use NLP datasets (e.g., GLUE, SQuAD) with streaming and preprocessing.
- **Experiment tracking** (e.g., MLflow, Weights & Biases): optional, for larger student projects.

# Evaluation in Practice

- Use train/validation/test splits managed with Pandas and scikit-learn.
- Compute accuracy, F1, confusion matrices and plot learning curves.
- Log metrics and configurations to reproduce experiments later.

# LLM & Prompt Tools

- **Hugging Face chat / inference UIs**: interact with open LLMs and test prompts.
- **Prompt playgrounds** (e.g., Agenta Prompt Playground): design, test, and compare prompts across models.
- **LangChain-compatible tools** (optional): build retrieval-augmented generation (RAG) and simple agents.

# Using LLM Tools in a Course

- Labs on prompt engineering: zero-shot, few-shot, and chain-of-thought prompts.
- Compare outputs from different LLMs for the same task and prompt.
- Explore safety filters and failure cases (bias, hallucinations, prompt injection).

# Recommended Minimal Toolset

- Core: Python, JupyterLab, NLTK, spaCy, scikit-learn, Gensim, Pandas, Matplotlib.
- Modern NLP: Hugging Face Transformers + Datasets, Sentence-Transformers, PyTorch.
- LLM practice: at least one prompt playground or LLM chat UI for hands-on experiments.

# From Features to Representations

- Limitations of sparse, hand-crafted features in capturing semantics.
- Distributional hypothesis: words that appear in similar contexts tend to have similar meanings.
- Dense vector representations (embeddings) encode semantic similarity.

# Word Embeddings

- word2vec, GloVe, and fastText map each word to a continuous vector.
- Semantic relationships captured via geometry (similarity, analogies).
- These embeddings are often pretrained on large corpora and reused across tasks.

# Sequence Models for NLP

- Many NLP tasks involve sequences: sentences, documents, conversations.
- Recurrent Neural Networks (RNNs), LSTMs, and GRUs process text token by token.
- Used for language modeling, sequence labeling, and simple sequence-to-sequence tasks.

# Core Neural NLP Tasks

- Text classification: sentiment analysis, topic classification, toxicity detection.
- Sequence labeling: POS tagging, named entity recognition (NER), chunking.
- Sequence-to-sequence: machine translation, summarization, question answering (QA).

# Limitations of RNN-based Models

- Difficulty capturing long-range dependencies in long sequences.
- Limited parallelism during training and inference due to sequential processing.
- Motivated the development of attention mechanisms and transformer architectures.

# Attention and Transformers

- Attention lets models focus on relevant parts of the input when generating an output.
- The Transformer architecture replaces recurrence with self-attention and feed-forward layers.
- Enables efficient parallel training and captures long-range dependencies effectively.

# Pretrained Transformer Models

- Encoder models: BERT, RoBERTa, DistilBERT for classification and sequence labeling.
- Decoder models: GPT-style models for text generation and dialogue.
- Encoder–decoder models: T5 and similar models for translation, summarization, and other seq2seq tasks.

# Large Language Models (LLMs)

- LLMs are transformer-based models trained on very large corpora with billions of parameters.
- Capable of general-purpose tasks: reasoning, coding, translation, summarization, and more.
- Enabled by advances in data, compute, and optimization techniques.

# Prompt Engineering

- Interacting with LLMs via prompts instead of training new models from scratch.
- Techniques: zero-shot, few-shot, and chain-of-thought prompting.
- Design principles: be specific, provide examples, and clearly define the desired output format.

# Examples of Prompts

- Classification-style prompts: "Classify the sentiment of this review as positive, neutral, or negative: ..."
- Instruction-style prompts: "Summarize the following article in three bullet points: ..."
- Structured-output prompts: "Extract all named entities (person, organization, location) from this text: ..."

# Safety and Responsible NLP

- Risks: biased outputs, toxic or harmful content, hallucinations, privacy leakage.
- Adversarial prompts and prompt injection can make models ignore original instructions.
- Mitigations: content filters, alignment techniques, guardrails, and human-in-the-loop oversight.