NLP Is All You Need

The Ultimate NLP Guide

Nichita Bulgaru, Timur Jerčaks

Maastricht, Netherlands
May 22, 2025

One small token for the model, one giant leap for understanding.

Contents

1 Introduction to Natural Language Processing (NLP)

1.1 What is NLP?

- **Natural Language Processing (NLP)**: A field of computer science, artificial intelligence, and linguistics concerned with the interactions between computers and human (natural) languages. It focuses on enabling computers to understand, interpret, generate, and process human language in a valuable way.
- **Relation to other fields**:
  - **Computational Linguistics (CL)**: Often used interchangeably with NLP, but CL can be more focused on the linguistic aspects and theories.
  - **Machine Learning (ML)**: Provides the methods/algorithms for NLP systems to learn from data. Many modern NLP approaches are data-driven.
  - **Deep Learning (DL)**: A subfield of ML using neural networks with multiple layers, which has led to significant breakthroughs in NLP.
- **Two Fundamental Questions in NLP**:
  1. How can computers *understand* and use natural language?
  2. To what extent can the properties of natural languages be *simulated computationally*?

1.2 Taskification in NLP

- **Taskification**: The process of converting a general question or problem into a well-defined NLP task that can be solved computationally (e.g., "How do I identify spam emails?" becomes "Build a spam/ham classifier").
- **Types of Tasks**:
  - **Intermediate Tasks (Tools/Building Blocks)**: Often serve as components for more complex applications.
    - Named Entity Recognition (NER)
    - Part-of-Speech (POS) Tagging
    - Parsing
    - Word Sense Disambiguation (WSD)
  - **Final Tasks (Products/Applications)**: Directly solve a user need.
    - Machine Translation (MT)
    - Text Summarization
    - Question Answering (QA)
    - Sentiment Analysis
    - Chatbots / Dialogue Systems
    - Information Retrieval (IR)

1.3 Key Challenges in Language Interpretation

- **Ambiguity**: The primary challenge. Language can be ambiguous at various levels:
  - *Lexical Ambiguity*: Words having multiple meanings (e.g., "bank" - financial institution or river bank; "can" - modal verb or container). This includes:
    - *Homonymy*: Words spelled and pronounced the same but with different unrelated meanings (e.g., "bat" - animal vs. sports equipment).
    - *Polysemy*: A single word with multiple related meanings (e.g., "bank" - financial institution, a place to store blood/data).
  - *Syntactic (Structural) Ambiguity*: A sentence can have multiple parse trees/interpretations (e.g., "I saw a man with a telescope").

  - *Phonetic Ambiguity*: Different phrases sounding similar (e.g., "I scream" vs. "ice cream").

- **Compositionality**: Understanding how the meaning of individual words/phrases combines to form the meaning of a larger text.
- **Co-reference Resolution**: Identifying all expressions in a text that refer to the same entity (e.g., "She said..." - who is "She"?).
- **Non-standard Language**: Slang, jargon, typos, grammatical errors, abbreviations prevalent in informal text (e.g., social media).
- **Neologisms & Language Evolution**: New words are constantly being created (e.g., "Brexit", "selfie"), and meanings of existing words change over time.
- **Rare Events (Zipf's Law)**: A few words are very frequent, but most words are rare. This leads to data sparsity issues.
- **Language Diversity**: Over 7000 languages exist, each with unique structures and complexities. NLP tools are predominantly developed for high-resource languages like English.
- **Scale**: The sheer volume of text data available.
- **World Knowledge / Common Sense**: Understanding language often requires knowledge beyond the text itself.

2 Approaches to NLP

2.1 Rule-based Approaches

- **Idea**: Manually define linguistic rules.
- **Examples**:
  - Grammars for parsing.
  - Lexicons with sentiment scores.
- **Pros**: Interpretable, can be precise for narrow domains.
- **Cons**:
  - Difficult to create and maintain comprehensive rules.
  - Do not scale well to new data or domains.
  - Struggle with ambiguity and language variability/evolution.
  - Brittle: fail on input not covered by rules.

2.2 Corpus-based / Statistical / Machine Learning Approaches

- **Idea**: Learn patterns and rules automatically from large collections of text (corpora).
- **Advantages**:
  - Can handle ambiguity and variability better.
  - Adaptable to new data/domains with retraining.
  - Often achieve higher performance on broad tasks.
- **Disadvantages**:
  - Require large amounts of (often annotated) data.
  - Models can be "black boxes", less interpretable.
  - Performance depends heavily on data quality and quantity.
- **Classical ML Pipeline**:

1. Data Collection & Preprocessing
2. Feature Engineering (manual design of input features, e.g., word counts, POS tags)
3. Model Training
4. Evaluation

2.3 Deep Learning Approaches

- **Idea**: Use neural networks with multiple layers to learn hierarchical representations (features) directly from raw text.
- **Key Concept - End-to-End Learning**: Often eliminates the need for manual feature engineering. The model learns the optimal representation for the task.
- **Drivers**:
  1. Advancements in hardware (GPUs).
  2. Availability of vast data.
  3. Improvements in neural network architectures and training techniques.
- **Examples**: Word embeddings (Word2Vec, GloVe, FastText), RNNs, LSTMs, Transformers (BERT, GPT).

3 Learning Scenarios in NLP

- **Supervised Learning**:
  - **Data**: Labeled data (input-output pairs, e.g., email → spam/not_spam).
  - **Goal**: Learn a mapping function from inputs to outputs.
  - **Examples**: Text classification, NER, MT (with parallel corpora).
- **Semi-supervised Learning**:
  - **Data**: A small amount of labeled data and a large amount of unlabeled data.
  - **Goal**: Leverage unlabeled data to improve supervised learning.
  - **Examples**: Using pre-trained word embeddings (trained on unlabeled data) as features for a supervised classifier.
- **Unsupervised Learning**:
  - **Data**: Unlabeled data.
  - **Goal**: Discover hidden structures or patterns in the data.
  - **Examples**: Topic modeling (LDA), clustering, word embedding training (e.g., Word2Vec).
- **Reinforcement Learning (RL)**:
  - **Data**: Interaction with an environment, receiving rewards or penalties.
  - **Goal**: Learn an optimal policy (sequence of actions) to maximize cumulative reward.
  - **Examples in NLP**: Dialogue systems (optimizing conversation flow), Reinforcement Learning from Human Feedback (RLHF) for aligning LLMs.

4 NLP Model Types

- **Classification**:
  - **Input**: Fixed-size representation (e.g., features of a word and its context).
  - **Output**: A single categorical label.
  - **Example**: Word Sense Disambiguation (predicting the correct sense of a word in context).
- **Sequence Classification**:
  - **Input**: A sequence of variable length (e.g., a sentence or document).
  - **Output**: A single categorical label for the entire sequence.
  - **Example**: Sentiment analysis (classifying a review as positive/negative), document classification.
- **Sequence Labeling (Tagging)**:
  - **Input**: A sequence of variable length.
  - **Output**: A sequence of labels of the same length as the input, where each label corresponds to an input token.
  - **Example**: Part-of-Speech (POS) tagging, Named Entity Recognition (NER).
- **Sequence-to-Sequence (Seq2Seq)**:
  - **Input**: A sequence of variable length.
  - **Output**: Another sequence of (potentially different) variable length.
  - **Example**: Machine Translation, Text Summarization, Question Answering (generative), Dialogue Generation.
- **Structured Prediction**:
  - **Input**: A sequence (or other structure).
  - **Output**: A complex structure (e.g., a tree, graph).
  - **Example**: Syntactic Parsing (generating a parse tree for a sentence).

5 Levels of Linguistic Analysis

- **Phonetics & Phonology**: Study of sounds.
  - *Phonetics*: Physical production and perception of speech sounds.
  - *Phonology*: Sound systems of languages, how sounds are organized and patterned (phonemes).
  - *Orthography*: Written representation of sounds.
- **Morphology**: Study of the internal structure of words.
  - *Morpheme*: Smallest meaningful unit in a language (e.g., "un-", "friend", "-ly", "-s").
  - Includes prefixes, suffixes, stems, roots.
  - Important for tasks like stemming, lemmatization, and handling unknown words in morphologically rich languages (e.g. Turkish, Finnish, German).
- **Syntax**: Study of the structure of sentences.
  - Grammatical relationships between words.
  - How words combine to form phrases and sentences (e.g., constituency parsing, dependency parsing).
  - Word order, grammatical categories (tense, voice, gender, number).
- **Semantics**: Study of meaning.

  - *Lexical Semantics*: Meaning of individual words (denotation, connotation, sense relations like synonymy, antonymy, hyponymy, hypernymy, meronymy).
  - *Compositional Semantics*: How the meanings of individual words combine to form the meaning of phrases and sentences.
  - Deals with polysemy, homonymy.
- **Pragmatics**: Study of language use in context.
  - How context (social, situational, linguistic) contributes to meaning.
  - Speaker's intention, implicatures, speech acts.
  - Example: "Can you pass the salt?" is a request, not just a question about ability.
- **Discourse**: Study of language units larger than a single sentence.
  - How sentences connect to form coherent and cohesive texts or conversations.
  - Cohesion (linguistic links: "however", pronouns), coherence (semantic links).
  - Narrative analysis, dialogue structure.

6 Text Preprocessing (Morphology, Tokenization, Normalization)

6.1 Morphology Basics

- **Morphology**: The study of the structure and formation of words.
- **Morpheme**: The smallest meaning-carrying unit in a language.
  - *Stem/Root Morphemes*: Core meaning unit (e.g., "house" in "houses").
  - *Affixes (Bound Morphemes)*: Attached to stems.
    - *Prefixes*: e.g., "un-" in "unhappy".
    - *Suffixes*: e.g., "-s" in "houses", "-est" in "smallest".
    - *Infixes*: Inserted within a stem (less common in English).
    - *Circumfixes*: Attached to both beginning and end.
- **Word Formation Processes**:
  - *Inflection*: Modifies a word's form to express grammatical features (tense, number, gender, case) without changing its core meaning or word class (e.g., "cat" → "cats", "walk" → "walked").
  - *Derivation*: Creates new words, often changing the word class or core meaning, by adding affixes (e.g., "happy" (adj) → "happiness" (noun), "predict" (verb) → "unpredictable" (adj)).
  - *Composition (Compounding)*: Combines two or more stem morphemes to create a new word (e.g., "rain" + "bow" → "rainbow", "note" + "book" → "notebook").
- **Morphological Analysis**: The process of identifying morphemes and word structure. Tools exist for well-resourced languages (e.g., `http://morphological.org/`).

6.2 Corpora: Tokens vs. Types

- **Corpus (pl. Corpora)**: A collection of natural language data (texts, speech).
  - *Raw Corpora*: Minimal or no processing (e.g., CommonCrawl).
  - *Annotated Corpora*: Contain labels or linguistic structures (e.g., Brown Corpus with POS tags).
- **Tokens**: All occurrences of words in a text (total word count). Example: "to be or not to be" has 6 tokens.

- **Types**: The set of unique words in a text (vocabulary size). Example: "to be or not to be" has 4 types: {"to", "be", "or", "not"}.
- **Vocabulary (V)**: The set of unique word types in a corpus.
- **Frequency (of a type)**: Number of times a type (word) appears as a token in a document/corpus.
- **Heap's Law**: Empirically observes that the vocabulary size ($|V|$) grows with the number of tokens (N) as $|V| = kN^{\beta}$, where $k > 0$ and $0 < \beta < 1$. Vocabulary size grows sub-linearly with corpus size.
- **Zipf's Law**: In a corpus, the frequency of any word is inversely proportional to its rank in the frequency table. A few words are very common, most words are rare (long tail distribution).

$$\text{word frequency} \propto \frac{1}{\text{word rank}}$$

  - *The good*: Common words provide structural cues.
  - *The bad*: Rare words have sparse data, making it hard to learn their properties.
  - *The ugly*: Unknown (Out-of-Vocabulary, OOV) words not seen in training.

6.3 Tokenization

Tokenization is the process of segmenting a sequence of characters into meaningful units called tokens (usually words, but can be subwords or characters).

6.3.1 Word Tokenization

- **Challenges in English**:
  - *Punctuation*: "home." → "home" , "." or "home."?
  - *Contractions*: "doesn't" → "does", "n't" or "does", "not"? or "doesn't"?
  - *Hyphenation*: "state-of-the-art" → one token or multiple?
  - *Abbreviations*: "D.C."
  - *Complex Names*: "New York", "COVID-19", "R2-D2".
  - *URLs, emails, numbers.*
- **Languages without spaces**: Chinese, Japanese, Thai require more complex word segmentation models, as words are not separated by spaces.
- **Agglutinative languages**: (e.g., Turkish, German) can form very long words by combining morphemes, e.g., German "Lebensversicherungsgesellschaftsangestellter".

Python Example (NLTK Word Tokenization):

```
import nltk
# nltk.download('punkt') # Run once if you haven't downloaded
from nltk.tokenize import word_tokenize

text = "Tokenization is an important first step. It's not always easy!"
tokens = word_tokenize(text)
print(tokens)
# Output: ['Tokenization', 'is', 'an', 'important', 'first', 'step', '.',
#          'It', "'s", 'not', 'always', 'easy', '!']
```

6.3.2 Sentence Segmentation (Sentence Tokenization)

The process of dividing a text into sentences.

- **Challenges**:
  - Periods (".") can mark end-of-sentence, abbreviations (e.g., "Inc.", "Dr."), or be part of numbers (e.g., ".02
  - Question marks ("?") and exclamation marks ("!") are less ambiguous.
- **Approaches**:
  - *Rule-based systems*: Using regular expressions or heuristics (e.g., period followed by space and capital letter).
  - *Machine Learning classifiers*: Train a model to decide if a punctuation mark is an end-of-sentence. Features could include capitalization, presence of numbers, type of word preceding/following.

Python Example (NLTK Sentence Tokenization):

```
import nltk
# nltk.download('punkt') # Run once
from nltk.tokenize import sent_tokenize

text = "Hello Dr. Smith. How are you today? I hope all is well."
sentences = sent_tokenize(text)
print(sentences)
# Output: ['Hello Dr. Smith.', 'How are you today?', 'I hope all is well.']
```

6.4 Text Normalization

The process of transforming text into a more uniform or canonical form to reduce data sparsity and improve consistency.

- **Goal**: Map several variant forms of a word to a single standard form.
- **Advantages**:
  - More examples of the same token (reduces vocabulary size).
  - Implicitly defines equivalence classes.
- **Task-dependent**: The level of normalization depends on the specific NLP task.

6.4.1 Case Folding / True-casing

- **Case Folding**: Converting all text to a single case (usually lowercase).
  - *Useful for*: Information Retrieval (search), as users often use lowercase.
  - *Potential Issues*: Can lose information if case is significant (e.g., "US" (United States) vs. "us" (pronoun), "Fed" (Federal Reserve) vs. "fed" (past tense of feed)). Proper nouns like "General Motors".
- **True-casing**: A more sophisticated approach that attempts to restore the correct casing for words (e.g., at the beginning of sentences or for proper nouns). Useful for MT, IE, sentiment analysis where case matters.

Python Example (Lowercase):

```
text = "The Quick Brown Fox Jumps Over The Lazy Dog."
lower_text = text.lower()
print(lower_text)
# Output: the quick brown fox jumps over the lazy dog.
```

6.4.2 Lemmatization

- **Definition**: Reducing inflected or variant forms of a word to its dictionary form, known as the **lemma**.
- Requires morphological analysis (understanding word structure and POS tags).
- **Examples**:
  - ”am”, ”are”, ”is” → ”be”
  - ”cars”, ”car’s”, ”cars’” → ”car”
  - ”running”, ”ran” → ”run” (if POS is verb)
- More linguistically principled than stemming.

Python Example (NLTK WordNet Lemmatizer):

```
import nltk
# nltk.download('wordnet') # Run once
# nltk.download('omw-1.4') # Run once for multilingual wordnet
# nltk.download('averaged_perceptron_tagger') # For POS tagging
from nltk.stem import WordNetLemmatizer
from nltk.corpus import wordnet # To map NLTK POS tags to WordNet POS tags

lemmatizer = WordNetLemmatizer()

def get_wordnet_pos(word):
    """Map POS tag to first character lemmatizer() accepts"""
    tag = nltk.pos_tag([word])[0][1][0].upper()
    tag_dict = {"J": wordnet.ADJ,
                "N": wordnet.NOUN,
                "V": wordnet.VERB,
                "R": wordnet.ADV}
    return tag_dict.get(tag, wordnet.NOUN) # Default to noun

word1 = "cars"
word2 = "running"
word3 = "better" # Adjective or Adverb

print(f"{word1} -> {lemmatizer.lemmatize(word1, get_wordnet_pos(word1))}")
# Output: cars -> car (if tagged as Noun)
print(f"{word2} -> {lemmatizer.lemmatize(word2, get_wordnet_pos(word2))}")
# Output: running -> run (if tagged as Verb)
print(f"{word3} -> {lemmatizer.lemmatize(word3, get_wordnet_pos(word3))}")
# Output: better -> good (if tagged as Adjective/Adverb)
# Note: 'better' can also be a verb 'to better something' -> 'better'
# lemmatizer.lemmatize('better', pos=wordnet.VERB) -> 'better'
```

6.4.3 Stemming

- **Definition**: A cruder heuristic process that chops off the ends of words (suffixes) to obtain a common ”stem”.
- Does not require morphological tools or knowledge of POS.
- Often faster than lemmatization.
- The resulting stem may not be a real word.
- **Examples** (Porter Stemmer):
  - ”automate(s)”, ”automatic”, ”automation” → ”automat”

– "computing", "computer", "computes" → "comput"

- Can lead to over-stemming (too much is cut off) or under-stemming (not enough is cut off).

Python Example (NLTK Porter Stemmer):

```
from nltk.stem import PorterStemmer

stemmer = PorterStemmer()
words = ["automate", "automates", "automatic", "automation", "computing"]
for word in words:
    print(f"{word} -> {stemmer.stem(word)}")
# Output:
# automate -> autom
# automates -> autom
# automatic -> automat
# automation -> autom
# computing -> comput
```

6.4.4 Summary of Normalization Techniques: Pros and Cons

Technique	What it does	When to use (with example)	When to avoid (with example)
Lowercasing	Converts all text to lowercase	Almost always – reduces vocabulary size *(e.g., spam detection, topic modeling)*	If case matters *(e.g., NER: `US` vs `us`, acronyms)*
Punctuation Removal	Removes punctuation symbols (e.g., '.' ',' '!')	For BoW/TF-IDF in classification *(e.g., document topic classification)*	If punctuation is meaningful *(e.g., sentiment analysis: "Great!" vs "Great.")*
Tokenization	Splits text into words or subwords	Always needed in any text model *(e.g., TF-IDF, BERT, translation)*	Rarely unnecessary *(only for raw text storage)*
Stopword Removal	Removes frequent functional words *(e.g., the, is, and)*	When topic matters *(e.g., topic classification)*	When style matters *(e.g., authorship attribution, writing style)*
Stemming	Cuts suffixes to get word stems *(e.g., running → run)*	For rough topic detection *(e.g., search engines)*	When morphology matters *(e.g., grammar correction)*
Lemmatization	Returns dictionary form *(e.g., better → good)*	When you need true word meaning *(e.g., machine translation)*	Slower and unnecessary in simple tasks *(e.g., binary spam classifier)*
Digit Removal / Replacement	Deletes or replaces numbers (e.g., `123` → `<num>`)	When numbers are not useful *(e.g., user ID, phone in spam)*	If numbers carry meaning *(e.g., price comparison, date detection)*
Spelling Correction	Fixes typos and spelling errors	For user-generated content *(e.g., Twitter sentiment)*	When spelling variation matters *(e.g., dialect studies)*

6.5 Handling Out-of-Vocabulary (OOV) Words

NLP systems often assume a fixed vocabulary learned from a training corpus. OOV words are those not present in this vocabulary.

- **Option 1: The `<UNK>` Token**:

  - Replace rare words in the training corpus (e.g., frequency $\leq$ threshold) with a special '¡UNK¿' (unknown) token.
  - During inference/testing, any word not in the vocabulary is mapped to '¡UNK¿'.
  - The model learns a representation for '¡UNK¿'.
  - Simple but loses information.

- **Option 2: Substring-based Representations**: (Common in Neural Models, LLMs)
  - Represent words as sequences of characters or character n-grams.
  - Allows the model to construct representations for OOV words based on their sub-parts.
  - Helps with misspellings and morphologically related words.
  - Example: Byte Pair Encoding (BPE).

6.6 Byte Pair Encoding (BPE) A data-driven tokenization algorithm that learns to segment words into subword units. It helps manage vocabulary size and handle OOV words.

- **Motivation**: Balances between character-level (too fine-grained) and word-level (large vocabulary, OOV issues) tokenization.
- **Core Idea**: Iteratively merge the most frequent pair of adjacent units (initially characters) in the training corpus to form new subword units.
- **Two Components**:
  1. **Token Learner (Vocabulary Induction)**:
     - Initialize vocabulary with all individual characters in the training corpus.
     - **Repeat k times (or until desired vocab size is reached)**:
       (a) Find the pair of adjacent symbols (e.g., 'A', 'B') that occurs most frequently in the training corpus.
       (b) Add the new merged symbol (e.g., 'AB') to the vocabulary.
       (c) Replace all occurrences of the pair ('A', 'B') in the corpus with the new symbol ('AB').
     - *Implementation Detail*: Often, words are first pre-processed by appending a special end-of-word symbol (e.g., `</w>` or `_`) and then split into characters. This helps distinguish subwords at the end of words from those in the middle. E.g., "newer" becomes `n e w e r _`.
  2. **Token Segmenter (Tokenization of New Text)**:
     - Given a new (test) sentence/word.
     - Apply the learned merge operations (from the token learner) greedily, in the order they were learned.
     - Test frequencies do not play a role in segmentation.
     - Example: If "e r" was merged to "er", and then "er _" was merged to "er_".
       * Test word `n e w e r _` $\rightarrow$ `n e w er_` $\rightarrow$ tokenized as `n`, `e`, `w`, `er_` (if `ne` or `ew` not merged earlier or more frequent). Or if `new` and `er_` are in vocab $\rightarrow$ `new`, `er_`. If `newer_` is in vocab $\rightarrow$ `newer_`.
       * Test word `l o w e r _` $\rightarrow$ could be `low`, `er_` if `low` and `er_` are learned tokens.
- **Properties of BPE Tokens**:
  - Vocabulary includes frequent words and frequent subwords (often corresponding to morphemes, e.g., "-est", "-er").
  - Achieves data compression.
  - Provides flexibility and handles open vocabularies (OOV words can be represented as sequences of known subwords).
  - Data-based approach (no explicit grammar knowledge needed).
  - Used in many Large Language Models (LLMs) like GPT.
  - *Criticism*: Tokenization choices can sometimes lead to suboptimal performance in LLMs on certain tasks (e.g., simple arithmetic, non-European languages if not trained well).

Conceptual BPE Learner Example (from slides): Corpus: 'low low low low low lowest lowest newer newer newer newer newer newer wider wider wider new new'

1. Add end-of-word token _: `l o w _` (5 times), `l o w e s t _` (2 times), `n e w e r _` (6 times), `w i d e r _` (3 times), `n e w _` (2 times)
2. Initial Vocab: _, `d`, `e`, `i`, `l`, `n`, `o`, `r`, `s`, `t`, `w`
3. Merge `e r` to `er` (most frequent pair, say). Corpus becomes e.g. `n e w er _`. Vocab: ..., `er`
4. Merge `er _` to `er_`. Corpus e.g. `n e w er_`. Vocab: ..., `er_`
5. Merge `n e` to `ne`. Corpus e.g. `ne w er_`. Vocab: ..., `ne`
6. And so on... next merges might be `(ne, w) -> new`, `(l, o) -> lo`, `(lo, w) -> low`, `(new, er_) -> newer_`, `(low, _) -> low_`.

Final vocabulary might include _, `d`, `e`, `i`, `l`, `n`, `o`, `r`, `s`, `t`, `w`, `er`, `er_`, `ne`, `new`, `lo`, `low`, `newer_`, `low_`, etc.

6.7 Delexicalization

Delexicalization is an NLP technique where specific, factual values in text are replaced with generic **placeholders** or **slot names**. For example, the original text "I see it costs $120, and rated four stars" becomes "I see it costs `[valuePrice]` and rated `[valueStars]` stars". This replaces specific values like `$120` and `four` with their corresponding generic placeholders.

6.7.1 Why is Delexicalization Used?

Delexicalization is employed to improve the **generalization**, **efficiency**, and **robustness** of NLP models, especially in dialogue systems:

1. **Generalization:** Allows models to learn general linguistic patterns and structures, rather than memorizing specific numerical or entity values. This improves performance on unseen data.
2. **Reduced Vocabulary Size:** Replacing an infinite number of specific values with a limited set of placeholders significantly reduces vocabulary size, leading to fewer model parameters, faster training, and lower memory consumption.
3. **Handling Out-of-Vocabulary (OOV) Issues:** Automatically handles new or out-of-vocabulary (OOV) specific values, as the model only needs to recognize the generic placeholder.
4. **Facilitates Dialogue System Components:** Particularly beneficial in task-oriented dialogue systems. It simplifies Dialogue State Tracking (extracting abstracted slot values) and Natural Language Generation (working with delexicalized templates before 'relexicalizing' with actual values).

In essence, delexicalization acts as an **abstraction layer**, allowing NLP models to learn more general linguistic patterns and handle a wider range of specific data points.

7 Basic Text Processing Tools

7.1 Regular Expressions (Regex)

A formal language for specifying text patterns. Widely used for pre-processing, searching, and manipulating text.

- **Basic Syntax Elements**:
  - `.` : Matches any single character (except newline).
  - `[]` : Disjunction - matches any single character within the brackets (e.g., `[abc]` matches 'a', 'b', or 'c').
  - `[^]` : Negation - matches any single character NOT within the brackets (when `^` is first char in `[]`, e.g., `[^0-9]` matches non-digits).
  - `-` : Range (within `[]`, e.g., `[a-z]` matches any lowercase letter, `[0-9]` matches any digit).
  - `|` : Alternation (OR operator, e.g., `cat`dog— matches "cat" or "dog").

  - `?` : Matches the preceding element zero or one time (optional). E.g., `colou?r` matches "color" or "colour".
  - `*` : Matches the preceding element zero or more times. E.g., `oo*h!` matches "oh!", "ooh!", "oooh!".
  - `+` : Matches the preceding element one or more times. E.g., `o+h!` matches "oh!", "ooh!", but not "h!".
  - `^` : Anchor - matches the beginning of a line (or string, depending on mode).
  - `$` : Anchor - matches the end of a line (or string).
  - `\b` : Matches a word boundary (position between a word character and a non-word character). E.g., `\bthe\b` matches "the" but not "other" or "they".
  - `\d` : Matches any digit (equivalent to `[0-9]`).
  - `\D` : Matches any non-digit.
  - `\w` : Matches any alphanumeric character (word character: letters, digits, underscore).
  - `\W` : Matches any non-word character.
  - `\s` : Matches any whitespace character (space, tab, newline).
  - `\S` : Matches any non-whitespace character.
  - `()` : Grouping. `(ab)+` matches "ab", "abab", etc.
  - `\` : Escape character (e.g., `\.` matches a literal dot, `\\` matches a literal backslash).

- **Example**: Finding all instances of "the" (case-insensitive, as a whole word): `\b[tT]he\b` or `(?i)\bthe\b` (using case-insensitive flag).
- **Example**: Give a regular expression that finds all occurrence of the word "go" and its morphological variants in a text that has not been preprocessed. `^ ][Gg]o(es|ing)?[$\. !?,]`

Python Example (re module):

```
import re

text = "The quick brown fox, the THE."
# Find all occurrences of "the" case-insensitive as a whole word
pattern = r"\bthe\b" # r"" denotes a raw string
matches = re.findall(pattern, text, re.IGNORECASE)
print(matches)
# Output: ['The', 'the', 'THE']

# Example: simple tokenization by splitting on non-alphanumeric
tokens_regex = re.split(r'\W+', text)
print([t for t in tokens_regex if t]) # Remove empty strings
# Output: ['The', 'quick', 'brown', 'fox', 'the', 'THE'] (punctuation lost)
```

7.2 Edit Distance (Levenshtein Distance)

The minimum number of single-character edits (insertions, deletions, or substitutions) required to change one word into the other.

- **Operations**:
  - *Insertion*: Add a character.
  - *Deletion*: Remove a character.
  - *Substitution*: Replace one character with another.
- **Cost**: Typically, each operation has a cost of 1. If substitutions cost 2 (and ins/del cost 1), it's specifically the Levenshtein distance definition.
- **Example**: 'INTENTION' → 'EXECUTION'
  - 'I N T E * N T I O N'

- '* E X E C U T I O N'
- d s s i s (d=delete, s=substitute, i=insert)
- If all cost 1, distance = 5 (del I, sub N/E, sub T/X, ins C, sub N/U).
- Or: del I, del N, del T, E-¿E, N-¿X, T-¿E, I-¿C, O-¿U, N-¿T, ins I, ins O, ins N (too many ops).
- Correct alignment for cost 1:

```
I N T E N T I O N
| | | |   | | | |
E X E C U T I O N
Costs:
I -> (del I) cost 1
N -> E (sub N/E) cost 1
T -> X (sub T/X) cost 1
E -> E (match) cost 0
N -> C (sub N/C) cost 1
T -> U (sub T/U) cost 1
I -> T (sub I/T) cost 1
O -> I (sub O/I) cost 1
N -> O (sub N/O) cost 1
(ins N) cost 1
This seems more like a general alignment problem.
Let's use standard example: INTENTION to EXECUTION
I N T E N T I O N
_ E X E C U T I O N  (Aligning for Levenshtein)
Operations:
1. Substitute I with E
2. Substitute N with X
3. Substitute T with E
4. Keep E
5. Substitute N with C
6. Substitute T with U
7. Keep T
8. Keep I
9. Keep O
10.Keep N
To transform INTENTION to EXECUTION:
Delete I (INTENTION -> NTENTION)
Delete N (NTENTION -> TENTION)
Delete T (TENTION -> ENTION)
E matches E
N -> X (substitute, ENTION -> EXTION)
T -> C (substitute, EXTION -> EXCION)
I -> U (substitute, EXCION -> EXCUON)
O matches O
N matches N
(This example in slide is for general alignment, not just Levenshtein)

Levenshtein distance (from slide example):
I N T E * N T I O N
* E X E C U T I O N
d s s   i s (delete, substitute, substitute, insert, substitute)
Cost = 5 if all ops cost 1.
Cost = 1(d) + 2(s) + 2(s) + 1(i) + 2(s) = 8 if sub costs 2.
```

- **Applications in NLP**:
  - *Spelling Correction*: Find words in dictionary closest to a misspelled word.

  - *Evaluating MT and ASR*: Word Error Rate (WER) is based on edit distance (but at the word level).
  - *Named Entity Recognition (NER) and Coreference Resolution*: Comparing strings for similarity.
  - *Computational Biology*: Aligning DNA/protein sequences.

- **Computation**: Typically computed using dynamic programming (e.g., Wagner-Fischer algorithm). A table $D(i, j)$ stores the edit distance between the first i characters of string X and the first j characters of string Y.

Python Example (using `nltk.edit_distance`):

```
import nltk

str1 = "intention"
str2 = "execution"

# NLTK's edit_distance assumes substitution cost is 1 by default
# if transpositions=False.
# If substitution_cost=2, it's Levenshtein.
# Default NLTK edit_distance: substitution costs 1 if chars are different, 0 if same.
# This is equivalent to Levenshtein if we define substitution cost as 1.
distance = nltk.edit_distance(str1, str2, substitution_cost=1, transpositions=False)
print(f"Edit distance between '{str1}' and '{str2}' (sub_cost=1): {distance}")
# Output: 5 (I->E, N->X, T->C, N->U, delete T. Or other sequence of 5 ops)
# Let's re-verify the slide's 5 ops:
# INTENTION -> ENTENTION (del I)
# ENTENTION -> EXTENTION (sub N by X)
# EXTENTION -> EXENTION (sub T by E - this is wrong, E from intention matches E from execution)
#
# Correct trace for Levenshtein (cost 1 for all):
# INTENTION
# EXECUTION
#
#   "" E X E C U T I O N
# "" 0  1 2 3 4 5 6 7 8 9
# I  1  1 2 3 4 5 6 7 8 9 (I!=E, sub I->E or del I + ins E)
# N  2  2 2 3 4 5 6 7 8 9 (N!=X, sub N->X)
# T  3  3 3 2 3 4 5 6 7 8 (T!=E, sub T->E)
# E  4  3 4 3 2 3 4 5 6 7 (E==E)
# N  5  4 3 4 3 2 3 4 5 6 (N!=C, sub N->C)
# T  6  5 4 3 4 3 2 3 4 5 (T!=U, sub T->U)
# I  7  6 5 4 3 4 3 2 3 4 (I==I)
# O  8  7 6 5 4 3 4 3 2 3 (O==O)
# N  9  8 7 6 5 4 3 4 3 2 (N==N)  The distance is 8.

# The slide example for INTENTION -> EXECUTION might use a different definition
# or visual alignment that's not strictly Levenshtein with cost 1 for all.
# The "d s s i s" implies 5 operations.
# I(del) N(sub E) T(sub X) E(match) *(ins C) N(sub U) T(match) I(match) O(match) N(match) -> 5 ops.
# This is the standard result for Levenshtein with unit costs.

# If substitution costs 2 (as per slide's Levenshtein condition for cost 8):
distance_lev = nltk.edit_distance(str1, str2, substitution_cost=2, transpositions=False)
print(f"Edit distance between '{str1}' and '{str2}' (sub_cost=2): {distance_lev}")
# Output: 8 (This matches the slide's Levenshtein example of 8)
# This means 3 substitutions (cost 3*2=6) and 2 indels (cost 2*1=2).
# Example sequence:
```

```
# INTENTION -> (del I) -> NTENTION (cost 1)
# NTENTION -> (del N) -> TENTION (cost 1)
# TENTION -> (sub T by E) -> EENTION (cost 2)
# EENTION -> (sub E by X) -> EXENTION (cost 2)
# EXENTION -> (sub N by C) -> EXECNTION (cost 2)
# Total 1+1+2+2+2 = 8. And EXECUTION has U, T, I, O, N. We have CNTION
# So 3 substitutions, and 2 deletions (I, N) to match the length of EXECUTION
# and then make the characters match. This is complex.

# Let's use a simpler library that is known for standard Levenshtein:
# pip install python-Levenshtein
# from Levenshtein import distance as lev_distance
# print(f"Levenshtein distance (python-Levenshtein): {lev_distance(str1, str2)}")
# Output: 5
# This confirms unit cost for substitution is standard for Levenshtein's original idea.
# The slide example with cost 8 implies substitution_cost=2 for Levenshtein definition.
```

Note on Edit Distance Example: The slide's calculation of 5 operations for INTENTION to EXECUTION assumes unit costs for insertion, deletion, and substitution. The calculation of 8 assumes substitutions cost 2, while insertions/deletions cost 1. NLTK's `edit_distance substitution_cost` parameter controls this.

8 Probabilistic Language Models: N-grams

8.1 Calculating Sentence Probability

The probability of a sequence of N words $P(w_1 w_2 \dots w_N)$ is calculated using the chain rule of probability:

$$P(w_1 w_2 \dots w_N) = \prod_{i=1}^{N} P(w_i | w_1 \dots w_{i-1})$$

Example: P("its water is so transparent") = P(its) $\times$ P(water|its) $\times$ P(is|its water) $\times \dots$

8.1.1 Markov Assumption

The history $w_1 \dots w_{i-1}$ is often too long. The Markov assumption simplifies this by assuming the probability of a word depends only on a fixed number of previous words (the context).

$$P(w_i | w_1 \dots w_{i-1}) \approx P(w_i | w_{i-n+1} \dots w_{i-1})$$

This leads to N-gram models.

8.2 Types of N-gram Models

- **Unigram Model (N=1):** Assumes words are independent.

  $$P(w_1 w_2 \dots w_N) = \prod_{i=1}^{N} P(w_i)$$

  Generated sentences are often incoherent (e.g., "Dog cat ball blue happy...").

- **Bigram Model (N=2):** Probability of a word depends on the previous word.

  $$P(w_1 w_2 \dots w_N) = \prod_{i=1}^{N} P(w_i | w_{i-1})$$

  (Conventionally, w_0 is a special start-of-sentence token $\langle s \rangle$). Generated sentences are more coherent (e.g., "The sun is shining brightly...").

- **Trigram Model (N=3), etc.:** Extends to trigrams, 4-grams, 5-grams. Generally, larger N captures more context but requires more data.

Raw Bigram Probabilities

- Normalize by unigrams (aka calculate the probabilities):

$$P(w_i|w_{i-1}) = \frac{\text{count}(w_{i-1}, w_i)}{\text{count}(w_{i-1})}$$

- Result:

Table 2: Unigram counts for w_{i-1}

	i	**want**	**to**	**eat**	**chinese**	**food**	**lunch**	**spend**
count(w_{i-1})	2533	927	2417	746	158	1093	341	278

Unigram Counts: count(w_{i-1})

Table 3: Calculated bigram probabilities $P(w_i|w_{i-1})$

$w_{i-1} \setminus w_i$	**i**	**want**	**to**	**eat**	**chinese**	**food**	**lunch**	**spend**
i	0.002	0.33	0	0.0036	0	0	0	0.00079
want	0.0022	0	0.66	0.0011	0.0065	0.0065	0.0054	0.0011
to	0.00083	0	0.0017	0.28	0.00083	0	0.0025	0.087
eat	0	0	0.0027	0	0.021	0.0027	0.056	0
chinese	0.0063	0	0	0	0	0.52	0.0063	0
food	0.014	0	0.014	0	0.00092	0.0037	0	0
lunch	0.0059	0	0	0	0	0.0029	0	0
spend	0.0036	0	0.0036	0	0	0	0	0

Bigram Probabilities: $P(w_i|w_{i-1})$

8.2.1 Limitations of N-gram Models

N-gram models struggle with **long-distance dependencies** (e.g., "The computer which I had just put into the machine room on the fifth floor crashed."). Modern Large Language Models (LLMs) handle this with much larger "context windows" (e.g., 8,000-32,000 tokens).

8.3 Training N-gram Language Models

8.3.1 Maximum Likelihood Estimation (MLE)

Parameters (N-gram probabilities) are trained to maximize the probability of the training data. For a bigram model, the MLE is:

$$P(w_i|w_{i-1}) = \frac{\text{count}(w_{i-1}, w_i)}{\text{count}(w_{i-1})}$$

The `count(wi-1, wi)` is the number of times the bigram appears, and `count(wi-1)` is the number of times the unigram w_{i-1} appears.

8.4 Evaluating Language Models

8.4.1 General Principles

- LMs should assign higher probability to "good" or "real" sentences than "bad" ones.
- Train on a **training set**.
- Evaluate performance on an unseen **test set** using an **evaluation metric**.

8.4.2 Extrinsic Evaluation

- Also called "down-stream" evaluation.
- Embed the LM into a task (e.g., spelling corrector, speech recognizer, MT system).
- Run the task and measure performance (e.g., accuracy).
- Compare performance of different LMs (A vs. B).
- **Challenge:** Can be time-consuming.

8.4.3 Intrinsic Evaluation: Perplexity (PP)

- **Intuition (Shannon Game):** How well can we predict the next word? A better model assigns higher probability to the word that actually occurs.
- **Definition:** Perplexity is the inverse probability of the test set, normalized by the number of words.
$$PP(W) = \sqrt[N]{\frac{1}{P(w_1 w_2 \dots w_N)}} = \left(\prod_{i=1}^{N} P(w_i | \text{context}_i) \right)^{-\frac{1}{N}}$$
where N is the number of words in the test set W.
- **Goal:** Lower perplexity is better. Minimizing perplexity is equivalent to maximizing the (log) probability of the test set.
- **Example (WSJ corpus):**
  - Unigram PP: 962
  - Bigram PP: 170
  - Trigram PP: 109
- **Note on LLMs:** It can be difficult to assess if the test set is truly unseen and not part of the massive training data of LLMs.

8.5 Example: Calculating Bigram Probabilities

Consider the following small training corpus (with start $\langle s \rangle$ and end $\langle /s \rangle$ tokens):

- $\langle s \rangle$ I am Sam $\langle /s \rangle$
- $\langle s \rangle$ Sam I am $\langle /s \rangle$
- $\langle s \rangle$ I do not like green eggs and ham $\langle /s \rangle$

Let's calculate some bigram probabilities:

1. $P(\text{I}|\langle s \rangle)$:
   - count($\langle s \rangle$, I) = 2 (appears in sentence 1 and 3)
   - count($\langle s \rangle$) = 3 (every sentence starts with $\langle s \rangle$)
   - $P(\text{I}|\langle s \rangle) = \frac{2}{3} \approx 0.67$
2. $P(\text{Sam}|\langle s \rangle)$:
   - count($\langle s \rangle$, Sam) = 1 (appears in sentence 2)
   - count($\langle s \rangle$) = 3
   - $P(\text{Sam}|\langle s \rangle) = \frac{1}{3} \approx 0.33$

This is how probabilities are derived from counts in the training data.

8.5.1 Example: Calculating Perplexity

Let's calculate the perplexity for a test sentence W = "I like green eggs"$\langle/s\rangle$ using a bigram model. Suppose our trained bigram model (after smoothing, if necessary) gives us the following probabilities:

- $P(\text{I}|\langle s\rangle) = 0.25$
- $P(\text{like}|\text{I}) = 0.30$
- $P(\text{green}|\text{like}) = 0.15$
- $P(\text{eggs}|\text{green}) = 0.40$
- $P(\langle/s\rangle|\text{eggs}) = 0.50$

First, calculate the probability of the sentence $P(W)$:

$$\begin{aligned} P(W) &= P(\text{I}|\langle s\rangle) \times P(\text{like}|\text{I}) \times P(\text{green}|\text{like}) \times P(\text{eggs}|\text{green}) \times P(\langle/s\rangle|\text{eggs}) \\ &= 0.25 \times 0.30 \times 0.15 \times 0.40 \times 0.50 \\ &= 0.075 \times 0.0075 \\ &= 0.00225 \end{aligned}$$

The number of words N in the test sentence (including $\langle/s\rangle$) is 5 ("I", "like", "green", "eggs", $\langle/s\rangle$). Now, calculate the perplexity:

$$\begin{aligned} PP(W) &= (P(W))^{-\frac{1}{N}} \\ &= (0.00225)^{-\frac{1}{5}} \\ &= \frac{1}{\sqrt[5]{0.00225}} \\ &\approx \frac{1}{0.3006} \\ &\approx 3.326 \end{aligned}$$

So, the perplexity of the sentence "I like green eggs $\langle/s\rangle$" with this bigram model is approximately 3.326. A lower perplexity would indicate a better fit of the model to this particular sentence.

8.6 Challenges and Practical Details

8.6.1 Overfitting

- With large vocabularies, the number of possible N-grams is huge (e.g., for Shakespeare, V=29,066, $V^2 \approx 844$ million possible bigrams, but only 300,000 observed).
- Many N-grams in a test set might not have been seen in training (zero count).
- Higher N-gram models are more prone to overfitting (memorizing the training data).

8.6.2 Zero Probabilities and Unseen N-grams

- If any N-gram in a test sentence has zero probability (because it was unseen in training), the entire sentence gets zero probability.
- This makes perplexity undefined (division by zero).
- **Solution:** Smoothing techniques.

Original:

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

Add-1:

	i	want	to	eat	chinese	food	lunch	spend
i	6	828	1	10	1	1	1	3
want	3	1	609	2	7	7	6	2
to	3	1	5	687	3	1	7	212
eat	1	1	3	1	17	3	43	1
chinese	2	1	1	1	1	83	2	1
food	16	1	16	1	2	5	1	1
lunch	3	1	1	1	1	2	1	1
spend	2	1	2	1	1	1	1	1

Figure 1: Add-1 smoothening example

8.6.3 Smoothing Techniques

Goal: Assign some non-zero probability mass to unseen N-grams by "stealing" it from seen N-grams.

- **Add-one (Laplace) Smoothing:** Add 1 to all N-gram counts. For bigrams:

$$P_{\text{Add-1}}(w_i|w_{i-1}) = \frac{\text{count}(w_{i-1}, w_i) + 1}{\text{count}(w_{i-1}) + |V|}$$

  where $|V|$ is the vocabulary size. This prevents zero probabilities but can sometimes over-discount frequent N-grams.

- **Interpolation:** Combine probabilities from different N-gram models (e.g., trigram, bigram, unigram). Example: $\lambda_1 P_3(w_i|w_{i-2}w_{i-1}) + \lambda_2 P_2(w_i|w_{i-1}) + \lambda_3 P_1(w_i)$, where $\sum \lambda_j = 1$.

- **Backoff:** Use higher-order N-gram if good evidence exists; otherwise, "back off" to a lower-order N-gram. Example: If trigram count for "Scottish beer drinkers" is 0, fall back to bigram probabilities of "Scottish beer" and "beer drinkers", then to unigrams.

8.6.4 Handling Out-of-Vocabulary (OOV) Words

- Words in the test set that were not in the training vocabulary.
- **Common Solution: Unknown Word Token (`<UNK>`)**
  1. Choose a vocabulary (fixed lexicon L).
  2. During training, replace words not in L with `<UNK>`.
  3. Train probabilities for `<UNK>` like any other word.
  4. At test time, map any OOV word to `<UNK>` and use its learned probabilities.
- **Alternative: Sub-word Representations** (e.g., Byte Pair Encoding - BPE). Breaks words into smaller units, reducing OOV issues.

8.6.5 Text Generation with LMs (Advanced)

- **Greedy Search:** At each step, choose the most likely next word: $w_k = \arg\max_{w \in V} P(w|w_1, \ldots, w_{k-1})$.
- **Sampling Methods:** Introduce randomness.
  - *Top-k sampling:* Sample from the k most probable next words.
  - *Top-p (nucleus) sampling:* Sample from the smallest set of words whose cumulative probability exceeds p.
- **Beam Search (more advanced):** Keep track of several top candidate sequences ("beams") at each step and extend them, "waiting" to evaluate longer sequences.

8.7 Summary of Key Concepts

- N-gram Language Models form the foundation for understanding text probability.
- Training involves MLE from counts.
- Evaluation uses perplexity (intrinsic) or task-based metrics (extrinsic).
- Smoothing is crucial for handling unseen N-grams and avoiding zero probabilities.
- OOV words are managed using techniques like `<UNK>` tokens or sub-word units.
- Neural LMs offer more powerful, context-aware modeling.

9 Information Retrieval (IR)

9.1 Definition and Goal

- **Information Retrieval (IR)**: The task of finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within large collections (usually stored on computers).
- **Goal**: Retrieve documents that are **relevant** to the user's query/information need.
- **Classic Search Model**: User Task → Information Need → Query → Search Engine → Results. Potential issues: user misconception, query misformulation.
- **Challenges**:
  - Scale and dynamic nature of data (e.g., web).
  - User query characteristics (short, ambiguous, typos, revisions).
  - Under/over-specified queries. (buying CDs" (money or music?))
  - Vague semantics of documents, need for inference.
  - Determining relevance is subjective (often low inter-annotator agreement).
- **Retrieval vs. Language Generation**:
  - *Retrieval (Search Engine)*: Selects existing documents/items from a library based on a query. E.g., $y = d_i$, where $i = \arg\max_k P(d_k|x)$.
  - *Generation (LLM like ChatGPT)*: Creates new text based on input. E.g., $y = G(x)$, where G is a complex function (model).

9.2 Boolean Retrieval Model

The simplest IR model based on set theory and Boolean algebra.

- **Representation**: Documents and queries represented as sets of terms.
- **Term-Document Incidence Matrix**: A binary matrix where rows represent terms (vocabulary) and columns represent documents. Entry (t, d) is 1 if term t is present in document d, and 0 otherwise.
- **Incidence Vector**: Each term corresponds to a binary vector (row in the matrix) indicating the documents it appears in.
- **Boolean Queries**: Queries formulated using terms and Boolean operators (AND, OR, NOT).
  - `term1 AND term2`: Intersection of document sets ($\cap$).
  - `term1 OR term2`: Union of document sets ($\cup$).
  - `NOT term1`: Complement of the document set.
- **Answering Queries**: Perform corresponding bitwise logical operations (AND, OR, NOT) on the incidence vectors of the query terms.

- **Pros**: Simple, predictable, precise (a document either matches or not).
- **Cons**:
  - *Exact Match*: Does not handle partial matches well. Difficult for users to formulate effective queries.
  - *No Ranking*: All matching documents are returned unordered. Does not reflect degree of relevance.
  - *Sparsity/Storage*: Term-document matrix is huge and very sparse.
  - Does not consider term frequency or importance.
  - Does not encode word order (not a LM)

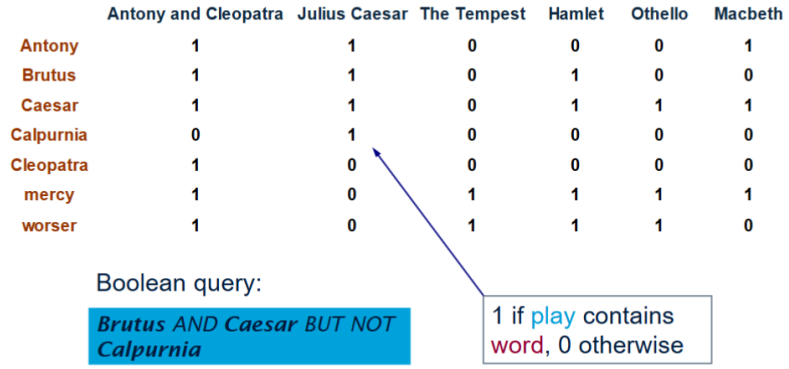


Figure 2: Term-document matrices

9.3 Inverted Index A more efficient data structure for IR, addressing the sparsity and storage issues of the term-document matrix.

- **Structure**: A dictionary (or similar structure) where keys are terms (vocabulary) and values are lists of document identifiers (and potentially positions) where the term appears. This list is called the **postings list**.
  - Example: 'Brutus' → [Doc1, Doc2, Doc4, ...]
- **Construction Pipeline**:
  1. **Collect Documents**: Gather the documents to be indexed. [Doc1: *"Friends, Romans, countrymen."*]
  2. **Tokenize Text**: Break documents into tokens (words). Doc1 → `["Friends", "Romans", "countrymen"]`
  3. **Linguistic Preprocessing**: Normalize tokens (e.g., case folding, stemming/lemmatization) and potentially remove stop words. [`"Friends"` → `"friends"` , `"countrymen"` → `"countryman"` , "their" (may be deleted)]
  4. **Index Creation**: Create the inverted index structure by associating each processed term with its postings list. Sort postings lists (usually by docID) for efficient merging. friend:[Doc1], countryman:[Doc1]...
- **Query Processing (Boolean)**:
  1. Retrieve postings lists for terms in the query.
  2. Perform set operations (intersection for AND, union for OR) on the postings lists. Typically involves efficient merge algorithms.
- **Advantages over Matrix**: Drastically reduces storage space by only storing the '1's from the sparse matrix. Allows fast retrieval of documents containing query terms. Logarithmic improvement in access. Many possible improvements/extensions(e.g. use bigrams).

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	157	73	0	0	0	0
Brutus	4	157	0	1	0	0
Caesar	232	227	0	2	1	1
Calpurnia	0	10	0	0	0	0
Cleopatra	57	0	0	0	0	0
mercy	2	0	3	5	5	1
worser	2	0	1	1	1	0

Figure 3: Each document is a count vector in $\mathbb{N}^{|V|}$: BOW

- **Disadvantages of the boolean models**: The model doesn't tell us how to choose or preprocess terms . It ignores how important a word is in a query or how well it represents a document. It returns documents as either matching or not — no scoring or ranking. Boolean matrices are mostly zeros; storing them fully is space-inefficient. Small changes in query can lead to very different or no results at all.

9.4 Good "basic concept"

The effectiveness of information retrieval heavily depends on the choice of basic concepts (terms). A good concept should be **orthogonal**—that is, semantically distinct and non-overlapping—allowing for clear representation and accurate matching between queries and documents. Ambiguity in terms (e.g., "Jaguar" meaning a car, an animal, or a music band) leads to poor retrieval quality due to confusion in meaning. Orthogonality ensures that each term contributes uniquely to the representation, and helps avoid misleading matches. Proper preprocessing and disambiguation are essential for improving search relevance and model performance.

9.5 Beyond Boolean: Ranked Retrieval

Boolean retrieval returns exact matches but doesn't rank them. Ranked retrieval models assign scores to documents based on their estimated relevance to the query.

- Query side: Not all terms are equally important
- Doc side: Some terms carry more information about content

9.6 Bag of words (BOW) model

The **Bag of Words (BoW)** Figure3 model represents text as a vector of word occurrences, completely ignoring the **order** in which words appear. As a result, sentences like *"John is quicker than Mary"* and *"Mary is quicker than John"* are treated identically, even though they convey different meanings. This simplification makes BoW easy to use and effective for tasks like **text classification**, but it also introduces limitations. Besides word order, BoW also struggles with issues such as **context loss** *(words are treated in isolation)*, **polysemy** *(a single word having multiple meanings)*, and **semantic similarity** *(different words with similar meanings are treated as unrelated)*, which can reduce its effectiveness in more nuanced language tasks.

9.7 Term Weighting: TF-IDF

Assigns weights to terms to indicate their importance in a document and across the collection.

9.7.1 Term Frequency (TF)

Term Frequency (TF) measures how often a term t appears in a document d.

- **Raw TF** ($tf_{t,d}$): the number of times term t occurs in document d.
- **Issue:** Relevance doesn't increase proportionally with raw frequency. For example, a term appearing 10 times is more relevant than one appearing once, but not necessarily 10 times more relevant.
- **Log-frequency weighting:** A common solution is to dampen the effect of high frequency using the formula:

$$w_{t,d} = \begin{cases} 1 + \log_{10}(tf_{t,d}) & \text{if } tf_{t,d} > 0 \\ 0 & \text{otherwise} \end{cases}$$

- **Alternative versions:** For example, augmented TF:

$$0.5 + 0.5 \cdot \frac{tf_{t,d}}{\max_t(tf_{t,d})}$$

The final score of a document can then be computed as:

$$\text{score} = \sum_{t \in q \cap d} (1 + \log tf_{t,d})$$

9.7.2 TF Normalization

Normalization is used to account for variation in document length. A long document may have more term occurrences simply because it is longer, not because it is more relevant.

- **Why normalize?**
  - Documents vary in length.
  - Repeated occurrences are less informative than the first.
- **Two perspectives on document length:**
  - A document is long because it uses more words.
  - A document is long because it contains more content.
- **General principle:** Penalize long documents to balance the comparison, but avoid over-penalizing.

Length Normalization with L_2 Norm

To normalize the vector of term frequencies, we divide each component by the L_2 norm of the vector:

$$\|\vec{x}\|_2 = \sqrt{\sum_i x_i^2}$$

Thus, normalized term frequency becomes:

$$tf_{t,d}^{\text{norm}} = \frac{tf_{t,d}}{\|\vec{x}\|_2}$$

This transformation produces a unit vector, ensuring that both long and short documents are comparable in terms of term frequency.

9.7.3 Inverse Document Frequency (IDF)

Measures how informative a term t is across the entire collection.

- **Document Frequency (df_t)**: The number of documents in the collection that contain term t. $df_t \leq N$, where N is the total number of documents.
- **Intuition**: Terms that appear in many documents (high df_t) are less informative than terms that appear in few documents (low df_t). Stop words have very high df_t.
- **IDF Formula**: Dampens the effect of df_t.

$$idf_t = \log_{10}\left(\frac{N}{df_t}\right)$$

(If $df_t = 0$, technically undefined, but often handled by adding 1 to denominator or using smoothing. If $df_t = N$, $idf_t = 0$).

- There is one IDF value per term in the vocabulary for a given collection.

9.7.4 TF-IDF Weight

Combines TF and IDF to get a composite weight for term t in document d.

- **Formula (Common variant)**:

$$w_{t,d} = tf.idf_{t,d} = (1 + \log_{10} tf_{t,d}) \times \log_{10}\left(\frac{N}{df_t}\right)$$

  (Using log-weighted TF). If $tf_{t,d} = 0$, then $w_{t,d} = 0$.

- **Properties**:
  - Highest weight for terms occurring frequently within a document but rarely in the rest of the collection.
  - Lower weight for terms occurring frequently across the collection (e.g., stop words, depending on processing).
  - Lower weight for terms occurring rarely within a document.
  - idf has no effect on ranking on the one term queries (needed at least two)
  - Stop words are excluded from TF-IDF calculations to reduce noise

- Score for the document based on the query:

$$\text{Score}(q, d) = \sum_{t \in q \cap d} \text{tf.idf}_{t,d}$$

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	5.25	3.18	0	0	0	0.35
Brutus	1.21	6.1	0	1	0	0
Caesar	8.59	2.54	0	1.51	0.25	0
Calpurnia	0	1.54	0	0	0	0
Cleopatra	2.85	0	0	0	0	0
mercy	1.51	0	1.9	0.12	5.25	0.88
worser	1.37	0	0.11	4.15	0.25	1.95

Figure 4: Each document is now represented by a real-valued vector of tf-idf weights

9.8 Example: TF-IDF Query: {apple}

We will compute the cosine similarity between the query and each document using the ltc.lnn weighting scheme.

Term	Doc1	Doc2
apple	3	0
phone	0	2
fruit	2	0

Table 4: Term-document count matrix

Step 1: Compute TF and TF weights

Query Term: apple

$$\text{Raw } \text{tf}_{\text{apple},q} = 1, \quad \text{tf-wt}_{\text{apple},q} = 1 + \log(1) = 1$$

—

Document 1:

$$\text{tf}_{\text{apple},d1} = 3, \quad \text{tf-wt}_{\text{apple},d1} = 1 + \log(3) = 1.477$$
$$\text{tf}_{\text{fruit},d1} = 2, \quad \text{tf-wt}_{\text{fruit},d1} = 1 + \log(2) = 1.301$$
$$\text{tf}_{\text{phone},d1} = 0 \Rightarrow \text{tf-wt}_{\text{phone},d1} = 0$$

Document 2:

$$\text{tf}_{\text{phone},d2} = 2, \quad \text{tf-wt}_{\text{phone},d2} = 1 + \log(2) = 1.301$$
$$\text{tf}_{\text{apple},d2} = \text{tf}_{\text{fruit},d2} = 0 \Rightarrow \text{tf-wt} = 0$$

Step 2: Compute IDF for each term

$$\text{df}_{\text{apple}} = 1, \quad \text{df}_{\text{fruit}} = 1, \quad \text{df}_{\text{phone}} = 1$$
$$\text{idf} = \log_{10}\left(\frac{(totalnumofdocs)}{df_t}\right) = \log_{10}\left(\frac{2}{df_t}\right) = \log_{10}(2) = 0.301$$

Step 3: Compute TF-IDF

Doc 1:

$$\text{tf-idf}_{\text{apple},d1} = 1.477 \cdot 0.301 = 0.445$$
$$\text{tf-idf}_{\text{fruit},d1} = 1.301 \cdot 0.301 = 0.392$$
$$\text{tf-idf}_{\text{phone},d1} = 0$$

Doc 2:

$$\text{tf-idf}_{\text{phone},d2} = 1.301 \cdot 0.301 = 0.392$$
$$\text{tf-idf}_{\text{apple},d2} = \text{tf-idf}_{\text{fruit},d2} = 0$$

Step 4: Normalize Document Vectors (L2 norm)

Doc 1:

$$\|d_1\| = \sqrt{0.445^2 + 0.392^2} = \sqrt{0.198 + 0.154} = \sqrt{0.352} \approx 0.593$$
$$\text{Normalized tf-idf}_{\text{apple},d1} = \frac{0.445}{0.593} = 0.750$$

Doc 2:

$$\|d_2\| = \sqrt{0.392^2} = 0.392, \quad \text{Normalized tf-idf}_{\text{phone},d2} = \frac{0.392}{0.392} = 1.0$$

Step 5: Compute Cosine Score with Query

$$\text{score}(q,d) = \sum_{t \in q}\left(\frac{tf_{t,q} \cdot idf_t}{\sqrt{\sum_{q_i \in q}(tf\text{-}idf(q_i,q))^2}} \cdot \frac{tf_{t,d} \cdot idf_t}{\sqrt{\sum_{d_i \in d}(tf\text{-}idf(d_i,d))^2}}\right)$$

Only the term **apple** is in the query and **simplified formula**, so:

$$\text{score}(q,d) = \sum_{t \in q} tf_{t,q} \cdot idf_t \cdot \frac{tf_{t,d} \cdot idf_t}{\sqrt{\sum_{d_i \in d}(tf\text{-}idf(d_i,d))^2}}$$

Doc 1:

$$\text{score}(q,d1) = \frac{1 \cdot 0.445}{\sqrt{1^2} \cdot \|d_1\|} = \frac{0.445}{0.593} \approx 0.750$$

Doc 2:

$$\text{score}(q,d2) = \frac{1 \cdot 0}{1 \cdot 0.392} = 0$$

Conclusion

Doc 1 is ranked higher for the query `apple`, since its score (0.750) is greater than Doc 2's score (0).

Python Example (Scikit-learn TF-IDF):

```
from sklearn.feature_extraction.text import TfidfVectorizer
import pandas as pd

corpus = [
    'this is the first document',
    'this document is the second document',
    'and this is the third one',
    'is this the first document',
]

vectorizer = TfidfVectorizer(use_idf=True, smooth_idf=True, norm='l2') # norm='l2' is default
tfidf_matrix = vectorizer.fit_transform(corpus)

# Get feature names (terms)
feature_names = vectorizer.get_feature_names_out()
# Display matrix (dense format for readability)
df_tfidf = pd.DataFrame(tfidf_matrix.toarray(), columns=feature_names)
print(df_tfidf)

# Note: Scikit-learn's TF-IDF calculation might differ slightly from the
# simple log formulas above (e.g., uses natural log, different smoothing/normalization).
# TF = raw count
# IDF = log( (N+1)/(df+1) ) + 1  (default smooth_idf=True)
# Then L2 normalization is applied to each document vector.
```

9.9 Vector Space Model (VSM) Represents documents and queries as vectors in a high-dimensional space, where dimensions correspond to terms. Relevance is calculated based on vector proximity.

- **Representation**:
  - Each document d is a vector $\vec{d} = (w_{1,d}, w_{2,d}, ..., w_{|V|,d})$, where $w_{t,d}$ is the TF-IDF weight (or other weight) of term t in document d.
  - Queries q are also represented as vectors $\vec{q}$. Often weighted similarly to documents (e.g., using TF-IDF weights of terms in the query).
- **Space Properties**:
  - Dimensions = $|V|$ (vocabulary size). Can be very large (millions).
  - Vectors are usually very sparse (most entries are zero).
- **Similarity Measure**: Proximity between query vector $\vec{q}$ and document vector $\vec{d}$ indicates relevance.
  - **Cosine Similarity** is commonly used: Measures the cosine of the angle between two vectors. It is independent of vector magnitude (document length).

$$\cos(\vec{q}, \vec{d}) = \frac{\vec{q} \cdot \vec{d}}{||\vec{q}||||\vec{d}||} = \frac{\sum_{t \in q \cap d} w_{t,q} \cdot w_{t,d}}{\sqrt{\sum_{t=1}^{|V|} w_{t,q}^2} \sqrt{\sum_{t=1}^{|V|} w_{t,d}^2}}$$

  - Ranges from -1 (opposite) to 1 (identical direction). For non-negative TF-IDF weights, range is 0 to 1.

  - If vectors are **L2-normalized** (unit length), cosine similarity simplifies to the dot product: $\cos(\vec{q}, \vec{d}) = \vec{q} \cdot \vec{d} = \sum_{t \in q \cap d} w_{t,q} \cdot w_{t,d}$.
  - Euclidean distance is usually inappropriate due to sensitivity to vector magnitude (document length).

- **Ranking**: Documents are ranked by their cosine similarity score with the query vector.

	Antony and Cleopatra	Julius Caesar	Hamlet	Macbeth
Antony	5.25	3.18	0	0.35
Brutus	1.21	6.1	1	0

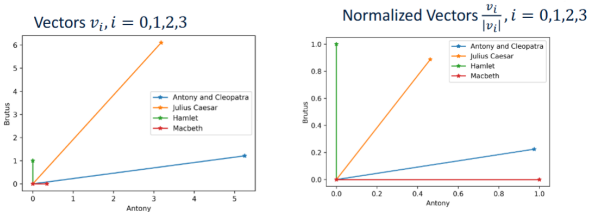


Figure 5: Tf-idf document vectors in the vector space

How similar are the novels SaS: *Sense and Sensibility* PaP: *Pride and Prejudice*, and WH: *Wuthering Heights*?

term	SaS	PaP	WH
affection	115	58	20
jealous	10	7	11
gossip	2	0	6
wuthering	0	0	38

Term frequencies (counts, tf)

Note: To simplify this example, we don't do idf weighting.

Figure 6: Cosine similarity amongst 3 documents

Cosine similarity amongst 3 documents, cont.

- Log frequency weighting (1+log(tf))

term	SaS	PaP	WH
affection	3.06	2.76	2.30
jealous	2.00	1.85	2.04
gossip	1.30	0	1.78
wuthering	0	0	2.58

- After length normalization

term	SaS	PaP	WH
affection	0.789	0.832	0.524
jealous	0.515	0.555	0.465
gossip	0.335	0	0.405
wuthering	0	0	0.588

cos(SaS,PaP) ≈
0.789 × 0.832 + 0.515 × 0.555 + 0.335 × 0.0 + 0.0 × 0.0 ≈ 0.94
cos(SaS,WH) ≈ 0.79
cos(PaP,WH) ≈ 0.69

Figure 7: Cosine similarity amongst 3 documents

Python Example (Cosine Similarity with Scikit-learn):

```
from sklearn.metrics.pairwise import cosine_similarity

# Assuming tfidf_matrix from previous example
# Example query: "this is the second"
query_vec = vectorizer.transform(['this is the second'])

# Calculate cosine similarity between query and all documents
```

```
cosine_similarities = cosine_similarity(query_vec, tfidf_matrix)
print(cosine_similarities)
# Output: [[0.5983... 0.8171... 0.5118... 0.5983...]]
# Document 1 (index 1) has highest similarity to the query.
```

9.10 Okapi BM25 A popular and effective probabilistic retrieval model, often outperforming standard TF-IDF VSM.

- **Goal**: Rank documents based on query terms, considering term frequency (non-linearly) and document length.
- **Formula (simplified view)**: Score is a sum over query terms $i \in q$.

$$Score(q,d) = \sum_{i \in q} IDF(q_i) \cdot \frac{tf_{i,d} \cdot (k_1 + 1)}{tf_{i,d} + k_1 \cdot \left(1 - b + b \cdot \frac{|d|}{avgdl}\right)}$$

  - $IDF(q_i)$: IDF weight of query term q_i.
  - $tf_{i,d}$: Term frequency of q_i in document d.
  - $|d|$: Length of document d.
  - $avgdl$: Average document length in the collection.
  - k_1: Parameter controlling term frequency saturation (typically 1.2-2.0). Higher k_1 means TF effect saturates later. $k_1 = 0$ gives a binary model.
  - b: Parameter controlling document length normalization (typically 0.75). $b = 0$ means no length normalization, $b = 1$ means full normalization relative to average length.
- **Advantages**: Strong empirical performance, well-regarded baseline. Better handling of TF saturation and document length normalization than basic TF-IDF.

9.11 Evaluation Metrics for IR How to measure the performance of an IR system? Requires a benchmark collection.

- **Benchmark Components**:
  1. Document Collection.
  2. Set of Information Needs (Queries).
  3. Relevance Judgments: For each query-document pair, a judgment (usually binary: relevant/not relevant) made by human assessors.
- **Unranked Retrieval (Set-based) Metrics**:
  - **Precision (P)**: Fraction of retrieved documents that are relevant. $P = \frac{|Relevant \cap Retrieved|}{|Retrieved|}$. Measures exactness. TP / (TP + FP)
  - **Recall (R)**: Fraction of relevant documents that are retrieved. $R = \frac{|Relevant \cap Retrieved|}{|Relevant|}$. Measures completeness. TP / (TP + FN)
  - **F-measure (F1 score)**: Harmonic mean of Precision and Recall. $F1 = \frac{2PR}{P+R}$. Balances P and R.
  - *Contingency Table*: Helps visualize True Positives (Relevant Retrieved), False Positives (Irrelevant Retrieved), False Negatives (Relevant Not Retrieved), True Negatives (Irrelevant Not Retrieved).
- **Ranked Retrieval Metrics**: Evaluate the ordering of results.
  - **Precision@K (P@K)**: Precision within the top K retrieved documents. Easy to compute, focuses on top results.
  - **Recall@K (R@K)**: Recall within the top K retrieved documents.
  - **Precision-Recall Curve**: Plots Precision against Recall at various rank thresholds (or recall levels). Shows the trade-off. Systems closer to the top-right corner are better.

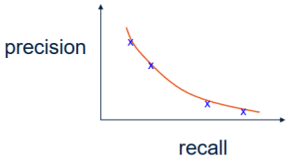


Figure 8: precision-recall-curve

- **R-Precision**: Precision at rank R, where R is the total number of known relevant documents for the query. $RPrec = \frac{\#\text{ relevant docs in top R}}{R}$.
- **Average Precision (AP)**: Average of precision values calculated each time a relevant document is found in the ranked list. Rewards systems that retrieve relevant documents early.

$$AP = \frac{\sum_{k=1}^{n}(P(k) \times rel(k))}{\text{number of relevant documents}}$$

where $P(k)$ is Precision@k, $rel(k)$ is 1 if item at rank k is relevant, 0 otherwise. n is total retrieved.

- **Mean Average Precision (MAP)**: The mean of AP scores across a set of queries. A single-figure measure widely used for system comparison. Macro-average, treats each query equally.

Evaluation Metrics: Examples

Retrieved results: T, T, F, F, T, F
(T = relevant, F = non-relevant)

Total relevant documents: 3

- **Precision@3** = $\frac{2}{3} \approx 0.667$ (True in top 3, from retrieved)
- **Recall@3** = $\frac{2}{3} \approx 0.667$ (True in top 3, from all)
- **R-Precision** = $\frac{2}{3} \approx 0.667$ (since R = 3, total True documents)
- **Average Precision (AP)** = $\frac{1+1+0.6}{3} \approx 0.867$
  (True docs at rank 1, 2, and 5 with P@1 = 1.0, P@2 = 1.0, P@5 = 0.6)
- **MAP** = AP = 0.867 (only one query)

Precision-Recall Table:

K	P@K	R@K
1	1.00	0.33
2	1.00	0.67
3	0.667	0.67
4	0.50	0.67
5	0.60	1.00

Python Example (Precision, Recall, F1):

```
from sklearn.metrics import precision_score, recall_score, f1_score

# Example scenario:
# True relevant documents for a query: {d1, d2, d4, d8} (Size R=4)
# System retrieves: {d1, d2, d3, d4, d5, d6, d7, d8, d9} (Total retrieved=9)
# Ranking: d1, d2, d3, d4, d5, d6, d7, d8, d9

# For the whole set retrieved (Unranked evaluation view):
retrieved_relevant = {d1, d2, d4, d8}
system_retrieved = {d1, d2, d3, d4, d5, d6, d7, d8, d9}
```

```
TP = len(retrieved_relevant) # 4
FP = len(system_retrieved - retrieved_relevant) # 5 (d3, d5, d6, d7, d9)
# Relevant docs: {d1, d2, d4, d8}
FN = 0 # All relevant were retrieved in this case

precision = TP / (TP + FP) # 4 / (4 + 5) = 4/9
recall = TP / (TP + FN + len(retrieved_relevant)) # 4 / 4 = 1.0 (assuming retrieved_relevant contain
# Let's assume true relevant set is R = {d1, d2, d4, d8, d10}. FN = 1 ({d10})
# Recall = TP / (TP + FN) = 4 / (4 + 1) = 0.8

# For simplicity, let's represent with lists of 0/1 (relevant/not relevant)
y_true = [1, 1, 0, 1, 0, 0, 0, 1, 0] # Relevance for d1..d9
y_pred = [1, 1, 1, 1, 1, 1, 1, 1, 1] # System retrieved all d1..d9

precision_sklearn = precision_score(y_true, y_pred) # For binary classification view
recall_sklearn = recall_score(y_true, y_pred)
f1_sklearn = f1_score(y_true, y_pred)

print(f"Precision: {precision_sklearn:.4f}") # 4/9 = 0.4444
print(f"Recall: {recall_sklearn:.4f}")    # 4/4 = 1.0 (if these are ALL relevant docs)
print(f"F1 Score: {f1_sklearn:.4f}")    # 2 * (0.4444 * 1.0) / (0.4444 + 1.0) = 0.6154

# Ranked Metrics Example:
# Ranking: d1(R), d2(R), d3(N), d4(R), d5(N), d6(N), d7(N), d8(R), d9(N)
# Relevant set size R = 4

# P@3 = 2/3 = 0.67
# P@5 = 3/5 = 0.60
# R-Precision (P@R = P@4) = 3/4 = 0.75

# Average Precision (AP):
# Precisions at relevant docs: P@1=1/1, P@2=2/2, P@4=3/4, P@8=4/8
AP = (1.0 + 1.0 + 0.75 + 0.5) / 4
print(f"Average Precision (AP): {AP:.4f}") # 0.8125
```

9.12 Addressing Bias in Information Retrieval Bias in Information Retrieval (IR) is a complex and unresolved issue. If I were the CEO of a search engine company aiming to reduce bias, I would consider the following strategies to modify the ranking algorithm and search pipeline:

1. **Incorporate fairness-aware ranking:** Modify scoring functions to balance relevance with fairness objectives, such as ensuring diversity of viewpoints or representation of marginalized groups.
2. **Query classification:** Detect potentially biased or opinionated queries (e.g., politically charged, gendered, or controversial topics) using natural language processing techniques. For such queries, apply debiasing strategies in document selection or reranking.
3. **Debiasing training data:** Ensure that training data for ranking models does not reinforce existing stereotypes or disproportionately favor dominant narratives.
4. **Audit and interpret model outputs:** Regularly analyze retrieval results for systemic bias. Tools such as fairness metrics and explainable AI (XAI) can help understand and mitigate problematic patterns.
5. **Include user feedback:** Allow users to report biased results, and integrate such feedback into system evaluation and model updates.
6. **Metadata-aware ranking:** Factor in document-level metadata (e.g., source credibility, publication region, authorship) to reduce the overrepresentation of dominant voices.

Bias cannot be eliminated entirely, but by recognizing its presence and designing with awareness, we can build more equitable and transparent retrieval systems.

10 Text Classification

10.1 Introduction and Applications

Text classification is the task of assigning predefined categories or labels to text documents.

- **Goal**: Assign one or more labels to a given text input (document, sentence, etc.).
- **Examples**:
  - Spam detection (spam/not spam)
  - Sentiment analysis (positive/negative/neutral)
  - Topic identification (sports/politics/technology)
  - Language identification (English/Spanish/German)
  - Authorship attribution
  - Age/gender identification
- **Sequence Classification**: A common formulation where the input is a sequence of variable length, and the output is a single label (or multiple labels for multi-label classification).

10.2 Challenges in Classification Tasks

- **Need for labeled data** - Requires input text and human-assigned labels.
- **Handling sequential inputs** - Sequences can have arbitrary length. There are dependencies within the sequence.
- **High dimensionality** - Vocabulary or feature space can be extremely large.
- **Ambiguity in label definitions** - Often, there is no single correct way to define labels.

10.3 NLP Datasets

- NLP models require **annotated (labeled) data** for training, validation, and evaluation.
- Annotation usually includes:
  - Input: typically text
  - Output: human-generated labels (e.g., sentiment, entity types)
- Labels may not always reflect *ground truth*, but serve as the **gold standard**.

10.3.1 Dataset Development Methods

1. **Paid, trained annotators** (e.g., linguists)
   - Example: Penn Treebank (1993)
   - High quality but costly
2. **Crowdsourcing** (e.g., Amazon Mechanical Turk)
   - Easier to scale, but lower control over annotator skill
   - Example: Stanford Sentiment Treebank
3. **Naturally-occurring annotation**
   - Data derived from interaction logs or usage (e.g., Alexa commands, reCAPTCHA)
   - Used in summarization, dialogue systems, etc.

10.3.2 Annotation Guidelines

- Define clear labeling rules and tag definitions.
- Handle ambiguous cases and gray zones (Several correct interpretations are possible).
- Specify annotation scope and tool usage.

10.3.3 Annotator Agreement and Cohen's Kappa

	Annotator A: Puppy	**Annotator A: Fried Chicken**
Annotator B: Puppy	6	3
Annotator B: Fried Chicken	2	5

Table 5: Confusion matrix: inter-annotator labeling agreement

Percent agreement = p_o = correct agreements / total items =

$$\frac{6+5}{6+3+2+5} \tag{1}$$

= 0.688

Cohen's Kappa accounts for chance agreement:

$$\kappa = \frac{p_o - p_e}{1 - p_e}$$

Where:

- p_o: observed agreement (e.g., 11/16 = 0.688)
- p_e: expected agreement by chance (e.g., 0.5)

$$\kappa = \frac{0.688 - 0.5}{1 - 0.5} = 0.376$$

10.3.4 Kappa Interpretation Guide

- 0.80–1.00: Very good agreement
- 0.60–0.80: Good agreement
- 0.40–0.60: Moderate agreement
- 0.20–0.40: Fair agreement
- ¡ 0.20: Poor agreement

10.4 Rule-based Text Classification Examples:

- **IF** there exists word w in document d such that w is in *[good, great, extraordinary, ...]*, **THEN** output Positive.
- **IF** email address ends in *[ithelpdesk.com, makemoney.com, spinthewheel.com, ...]*, **THEN** output SPAM.

Pros and Cons:

+ Can be very accurate (if rules are carefully refined by experts)
- Rules may be hard to define (and some unknown to us?)
- Expensive to build and maintain
- Not easily generalizable

10.5 Representing Text for Classification (BOW/Feature Engineering)

- **Bag-of-Words (BoW)**:
  - Represents text as a fixed-length vector where each dimension corresponds to a word in the vocabulary.
  - The value in each dimension is typically the count (or frequency) of that word in the document. TF-IDF weights can also be used.
  - **Ignores word order.**
  - Example: 'w = "The drinks were strong but the fish tacos were bland"' → 'x = [0, ..., 1 (bland), ..., 1 (but), ..., 1 (fish), ...]'.
  - Vector length = Vocabulary size ($|V|$). Can be very high dimensional.
- **Feature Engineering**: Manually designing features relevant to the task.
  - Can include BoW counts, TF-IDF weights.
  - Presence/absence of specific keywords (e.g., positive/negative words for sentiment).
  - N-grams (captures some local word order).
  - Text statistics (length, avg. word length, punctuation counts, capitalization).
  - Linguistic features (POS tags, dependency relations).
  - *Challenges*: Can be costly, time-consuming, domain-specific. Features might not generalize well. Ambiguity (e.g., "great" can be sarcastic).
- **Deep Learning Representations**: Word embeddings (Word2Vec, GloVe, FastText), sentence embeddings, contextual embeddings (BERT, ELMo) learned automatically (covered later).

10.6 What is a Feature Function?

In text classification, we often want a machine to decide whether some input (e.g., a sentence) is **positive**, **negative**, or something else.

To do this, we convert the input into numbers — these are called **features**. A **feature function** is what turns the input *and* a candidate label into a numerical feature vector.

Simple explanation:

- A feature function checks for things like specific words.
- It says: "If the label is 'negative' and the word 'bland' is in the sentence, turn on that feature."
- If you're checking for the label 'positive', that feature is turned off (zero), even if 'bland' is in the text.

10.6.1 Example Feature Function

$$f_j(x, y) = \begin{cases} x_{\text{bland}}, & \text{if } y = \text{negative} \\ 0, & \text{otherwise} \end{cases}$$

If we check the label "negative" and there is the word "bland" in the text - we switch on the feature. If we check "positive" - ignore it (reset it to zero).

10.6.2 Why do we do this?

- We want to compare each label (positive, negative, etc.) with the input.
- For each label, we build a separate feature vector.
- Combine features and label into a single vector.
- Then we see which label's feature vector "fits" the input best.
- The model was able to learn separate weights for each label.

10.6.3 Analogy:

It's like asking:

> "If I assume this sentence is **positive**, what does it look like? What about if it's **negative**? Which version makes more sense?"

The model answers: "Here is what this example looks like, assuming that the label is a y." (This is done by scoring each version using feature functions.)

10.6.4 Example: Vector Form for Multiple Labels

If we have K possible labels (e.g., `positive`, `negative`, `neutral`), the feature function creates **different versions of the input features for each possible label**.

- Each version activates the input vector only if the label matches.
- We have **the same feature structure** (same dictionary, order, vector position), no matter which label we check.
- For all other labels, the vector is filled with zeros.
- This creates a large feature vector of size $K \times V$, where V is the number of features per input.

$$f(x, y = 1) = \begin{bmatrix} x_0 & x_1 & \dots & x_M & 0 & 0 & \dots & 0 \end{bmatrix}^T$$
$$f(x, y = 2) = \begin{bmatrix} 0 & 0 & \dots & 0 & x_0 & x_1 & \dots & x_M & \dots \end{bmatrix}^T$$
$$f(x, y = K) = \begin{bmatrix} 0 & 0 & \dots & 0 & x_0 & x_1 & \dots & x_M \end{bmatrix}^T$$

This means: for each candidate label y, we "shift" the input vector x into the slot corresponding to y and fill the rest with zeros. This helps the model learn different weightings for the same input depending on the label. **Otherwise, all tags would "look" at the same attributes - and there would be no distinction between them.**

10.6.5 Feature Engineering Issues/Solutions

Issue	Example	Possible Solution
Word ambiguity: Word appears positive but isn't	*"It's not a **great** monster movie."* — negation flips the meaning	Use dependency parsing or syntactic analysis to detect negation
Different senses of the same word	*"There's a **great** deal of corny dialogue..."* — "great" means "large amount," not "good"	Use word sense disambiguation or contextual embeddings like BERT
Mixed or subtle sentiment	*"A **great** ensemble cast can't lift this heartfelt enterprise..."* — mixed positive and negative tones	Use models that handle sentence-level or phrase-level composition, e.g., RNNs or transformers
Literal vs. figurative meaning	Words like "killer" in slang can be positive: *"That was a killer performance."*	Use contextual models trained on in-domain data (e.g., social media)

10.7 Naive Bayes Classifier

A simple and effective probabilistic classifier based on Bayes' theorem with strong ("naive") independence assumptions. It's a **generative** model.

It answers the question:

> **What is the probability that this message belongs to class A (e.g., "spam")?**

To do this, it looks at:

- the **words** in the message,
- the **precomputed statistics** for each class (like spam / not spam),
- and it **estimates which class is more likely**.

Why "Naive"? - because it assumes that all words are **independent** of each other, which is not true in real language — but it still works surprisingly well!

- **Goal**: Find the most likely class c for a given document d.
- **MAP Estimate**: $\hat{c}_{MAP} = \arg\max_{c \in C} P(c|d)$.
- **Using Bayes' Theorem**:
$$P(c|d) = \frac{P(d|c)P(c)}{P(d)}$$
$$\hat{c}_{MAP} = \arg\max_{c \in C} P(d|c)P(c)$$
($P(d)$ is dropped as it's constant for all classes).
- **Representation**: Document d is represented as a set of features $X = \{x_1, x_2, ..., x_n\}$ (e.g., words in BoW).
$$\hat{c}_{MAP} = \arg\max_{c \in C} P(x_1, x_2, ..., x_n|c)P(c)$$
- **Naive Independence Assumption**: Features (words) are conditionally independent given the class.
$$P(x_1, x_2, ..., x_n|c) = \prod_{i=1}^{n} P(x_i|c)$$
- **Multinomial Naive Bayes Formula**: For BoW features (word counts). $pos(d)$ are the positions in the document.
$$\hat{c}_{NB} = \arg\max_{c \in C} P(c) \prod_{i \in positions(d)} P(word\_at(i)|c)$$
In practice, we often calculate in log space to avoid underflow:
$$\hat{c}_{NB} = \arg\max_{c \in C} \left(\log P(c) + \sum_{i \in positions(d)} \log P(word\_at(i)|c) \right)$$
- **Parameter Estimation (Training - MLE)**:
  - Prior probability of class c: $\hat{P}(c) = \frac{N_c}{N_{doc}}$ (Fraction of documents in class c).
  - Conditional probability of word w given class c:
$$\hat{P}(w|c) = \frac{count(w, c)}{\sum_{w' \in V} count(w', c)}$$
(Relative frequency of word w in all documents of class c).

Problem	Solution
Zero Probabilities: If a word w never appears in training documents of class c, then $\hat{P}(w\|c) = 0$. This causes the total probability of a test document to be zero for that class if w appears in it.	**Smoothing (Laplace / Add-one):** Pretend every word appeared at least once in each class. Formula: $$\hat{P}_{\text{Add-1}}(w|c) = \frac{\text{count}(w, c) + 1}{\sum_{w' \in V} \text{count}(w', c) + \|V\|}$$ Other variants include Add-k smoothing.

Table 7: Handling Zero Probabilities in Naive Bayes

Pros	Cons
Very fast and efficient	Assumes feature independence, which is often unrealistic
Low memory usage	May perform worse than discriminative models (e.g., logistic regression, SVM)
Robust to irrelevant features	Can't model complex interactions between features
Good baseline for text tasks	Performance drops with correlated features
Works well with Bag-of-Words (BoW)	Assumes all features contribute equally and independently

Table 8: Advantages and limitations of Naive Bayes classifier

10.7.1 Example: Spam Detection with Naive Bayes

Classify whether a given email is **spam** or **ham** (no spam) based on its words using a Naive Bayes classifier.

Step-by-step Collect training data:

- "Win a free iPhone now" → spam
- "Meeting schedule attached" → ham

Preprocess:
Convert messages to lowercase, tokenize, and build a vocabulary.
Count word frequencies:

Word	Count in Spam	Count in Ham
win	3	0
free	2	0
iPhone	1	0
meeting	0	2
schedule	0	1

The prior probability of a class (such as `spam` or `ham`) indicates how frequently that class occurs in the training data. These are computed as:

$$P(c) = \frac{\text{Number of documents in class } c}{\text{Total number of documents}}$$

For example, if we have 100 emails:

- 40 are labeled as `spam`
- 60 are labeled as `ham`

Then the prior probabilities are:

$$P(\text{spam}) = \frac{40}{100} = 0.4 \qquad P(\text{ham}) = \frac{60}{100} = 0.6$$

These priors are multiplied with the word likelihoods during classification, according to Bayes' Rule:

$$P(c \mid \text{words}) \propto P(c) \cdot \prod_{i=1}^{n} P(w_i \mid c)$$

In unbalanced cases, we use:

- class balancing (undersampling / oversampling),

- or just set equal probabilities manually

Estimate probabilities (with Laplace smoothing):

$$\hat{P}(w \mid \text{spam}) = \frac{\text{count}(w, \text{spam}) + 1}{\text{total words in spam} + |V|}$$

Classify a new message:
"free meeting now"
Compute both:

$$P(\text{spam} \mid \text{message}) \propto P(\text{spam}) \cdot P(\text{free} \mid \text{spam}) \cdot P(\text{meeting} \mid \text{spam}) \cdot P(\text{now} \mid \text{spam})$$

$$P(\text{ham} \mid \text{message}) \propto P(\text{ham}) \cdot P(\text{free} \mid \text{ham}) \cdot P(\text{meeting} \mid \text{ham}) \cdot P(\text{now} \mid \text{ham})$$

Pick the label with the higher probability.

Result

If $P(\text{spam} \mid \text{message}) > P(\text{ham} \mid \text{message})$, classify as `spam` — otherwise, as `ham`.

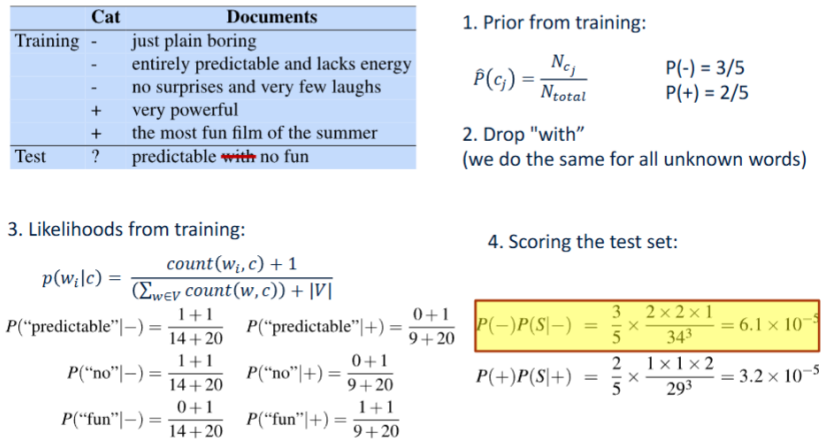


Figure 9: Ssentiment example with add-1 smoothing

Python Example (Scikit-learn MultinomialNB):

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.pipeline import make_pipeline
from sklearn.metrics import accuracy_score

# Example Data
corpus = [
    'this is a great great movie', # positive
    'this movie is terrible',      # negative
    'what a wonderful film',       # positive
    'I did not like this movie',   # negative
]
labels = ['positive', 'negative', 'positive', 'negative']

# Create a pipeline: CountVectorizer (BoW) + MultinomialNB
# CountVectorizer converts text to word count vectors.
# MultinomialNB implements Naive Bayes for count data.
# alpha=1.0 enables Laplace (add-one) smoothing.
model = make_pipeline(CountVectorizer(), MultinomialNB(alpha=1.0))
```

```
# Train the model
model.fit(corpus, labels)

# Predict on new data
test_data = ['this movie was great and wonderful']
prediction = model.predict(test_data)
print(f"Prediction for '{test_data[0]}': {prediction[0]}")
# Output: Prediction for 'this movie was great and wonderful': positive

# Predict probabilities
probabilities = model.predict_proba(test_data)
print(f"Probabilities: {model.classes_} -> {probabilities}")
# Output: Probabilities: ['negative' 'positive'] -> [[0.18... 0.81...]]
```

10.8 Logistic Regression

A **discriminative linear classifier** widely used for binary classification tasks.

- **Goal**: Directly model the posterior probability $P(y|x)$.
- **Linear Component**: Compute a score $z = w \cdot x + b = \sum_i w_i x_i + b$, where w are weights, x are features, and b is bias.
- **Sigmoid Function**: Maps the score z to a probability between 0 and 1.

$$P(y = 1|x) = \hat{y} = \sigma(z) = \frac{1}{1 + e^{-z}}$$

$$P(y = 0|x) = 1 - \sigma(z)$$

- **Decision Boundary**: Classify as positive if $P(y = 1|x) >$ threshold (often 0.5). This corresponds to $z > 0$ (if threshold is 0.5). The decision boundary $w \cdot x + b = 0$ is linear.
- **Features**: Can be BoW counts, TF-IDF values, or manually engineered features.
- **Learning**: Find weights w and bias b (parameters θ) that optimize performance on training data.
- **Loss Function: Cross-Entropy Loss** (Negative Log-Likelihood): Measures the difference between predicted probabilities and true labels (0 or 1). For a single instance (x, y):

$$L_{CE}(\hat{y}, y) = -[y \log \hat{y} + (1 - y) \log(1 - \hat{y})]$$

  The goal is to minimize the average cross-entropy loss over the training set:

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^{m} [y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})]$$

- **Optimization: Gradient Descent**: Iteratively update parameters $\theta = (w, b)$ in the opposite direction of the gradient of the loss function $J(\theta)$.

$$\theta_{new} = \theta_{old} - \eta \nabla_\theta J(\theta)$$

  where η is the learning rate.

  - *Batch GD*: Computes gradient over the entire training set.
  - *Stochastic GD (SGD)*: Computes gradient for one instance at a time (or a small mini-batch). Faster updates, can escape local minima (though LR loss is convex).

- **Softmax Regression**: Generalization for multi-class classification, using the softmax function instead of sigmoid in the output layer.

10.8.1 Features in Logistic Regression

In logistic regression, we need to convert text into numbers (features) so the model can use them.

Types of Features

- **Bag-of-Words (BoW):** Each word becomes a feature. The model sees how often each word appears.
- **Manual features:** You can create your own features like:
  - Number of positive or negative words
  - Sentence length
  - Use of specific keywords (e.g., "awesome", "boring")

10.8.2 Feature Example with Calculation

We want the model to predict:

$$y = \begin{cases} 1 & \text{if sentiment is positive} \\ 0 & \text{if sentiment is negative} \end{cases}$$

Example text:

> "It's hokey. There are virtually no surprises, and the writing is second-rate. So why was it so enjoyable? For one thing, the cast is great. Another nice touch is the music. I was overcome with the urge to get off the couch and start dancing. It sucked me in, and it'll do the same to you."

The model looks at the words (like `hokey`, `great`, `sucked`) and decides whether overall sentiment is positive or negative.

"Great", "nice", "dancing" → positive weights "Second-rate", "hokey" → negative weights
It adds up these word contributions and applies a decision rule to classify the text.

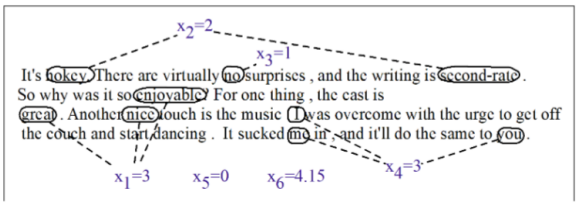


Figure 10: Features in logistic regression

We extract a set of features from the document. Let $x = [x_1, x_2, x_3, x_4, x_5, x_6]$ be the input vector.

Feature	Definition	Value
x_1	Count of positive words	3
x_2	Count of negative words	2
x_3	1 if word "no" appears, else 0	1
x_4	Count of 1st/2nd person pronouns	3
x_5	1 if "!" appears, else 0	0
x_6	log(number of words in doc)	4.15

Table 9: Features extracted from the input document

Assume learned weights:

$$w = [2.5,\ -5.0,\ -1.2,\ 0.5,\ 2.0,\ 0.7] \quad \text{and bias } b = 0.1$$

We compute:

$$z = w \cdot x + b = 2.5 \cdot 3 + (-5.0) \cdot 2 + (-1.2) \cdot 1 + 0.5 \cdot 3 + 2.0 \cdot 0 + 0.7 \cdot 4.15 + 0.1$$

$$z = 7.5 - 10 - 1.2 + 1.5 + 0 + 2.905 + 0.1 = 0.805$$

Then apply the sigmoid function:

$$\hat{y} = \sigma(z) = \frac{1}{1 + e^{-0.805}} \approx 0.691$$

So, the predicted probability of positive class (e.g., positive sentiment):

$$P(y = 1 \mid x) \approx 0.69 \quad \Rightarrow \quad \text{likely positive}$$

$$P(y = 0 \mid x) = 1 - \hat{y} \approx 0.31$$

10.8.3 Cross-Entropy Calculation

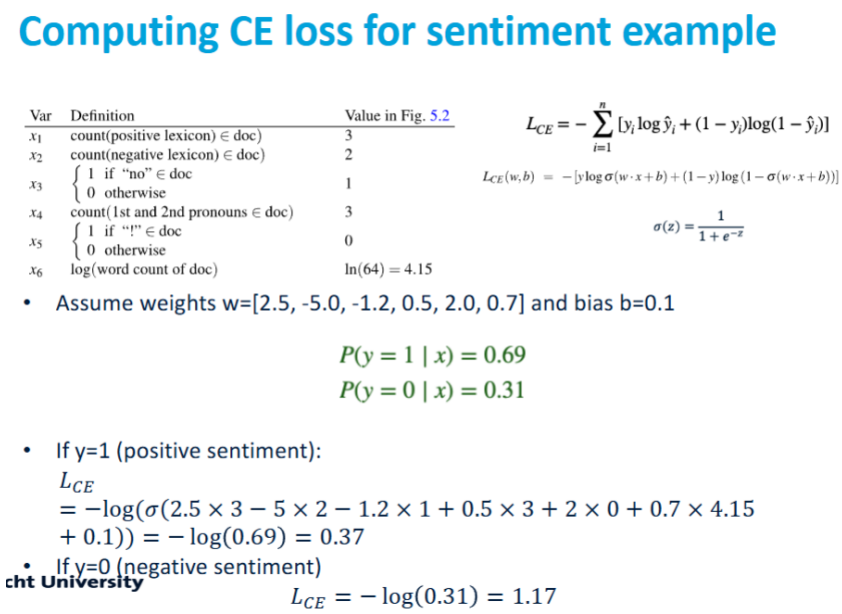


Figure 11: Computing Cross-Entropy loss for sentiment example

Python Example (Scikit-learn Logistic Regression):

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import make_pipeline

# Using same corpus and labels from NB example
corpus = [
    'this is a great great movie', # positive
    'this movie is terrible',      # negative
    'what a wonderful film',       # positive
    'I did not like this movie',   # negative
]
labels = ['positive', 'negative', 'positive', 'negative']
```

```
# Pipeline: TF-IDF Vectorizer + Logistic Regression
# TF-IDF is often preferred over raw counts for LR
model_lr = make_pipeline(TfidfVectorizer(), LogisticRegression(solver='liblinear'))

# Train
model_lr.fit(corpus, labels)

# Predict
test_data = ['this movie was great and wonderful']
prediction_lr = model_lr.predict(test_data)
print(f"LR Prediction for '{test_data[0]}': {prediction_lr[0]}")
# Output: LR Prediction for 'this movie was great and wonderful': positive

probabilities_lr = model_lr.predict_proba(test_data)
print(f"LR Probabilities: {model_lr.classes_} -> {probabilities_lr}")
# Output: LR Probabilities: ['negative' 'positive'] -> [[0.29... 0.70...]]
```

10.9 Evaluation Metrics for Classification Metrics beyond simple accuracy, especially important for imbalanced datasets.

- **Contingency Table (Confusion Matrix)** for a binary class:

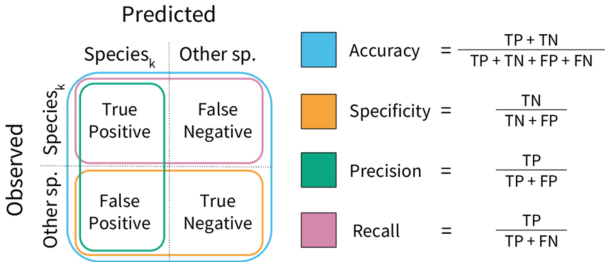


Figure 12: Contingency Table

- **Accuracy**: Overall correctness. $Acc = \frac{TP+TN}{TP+FP+FN+TN}$. Can be misleading in imbalanced datasets.
- **Precision (P)**: Of those predicted positive, how many actually are? $P = \frac{TP}{TP+FP}$. Measures exactness. High precision means low false positives.
- **Recall (R) / Sensitivity / True Positive Rate (TPR)**: Of those actually positive, how many were predicted? $R = \frac{TP}{TP+FN}$. Measures completeness. High recall means low false negatives.
- **F1-Score**: Harmonic mean of Precision and Recall. $F1 = \frac{2PR}{P+R}$. Useful when balancing P and R is needed.
- **False Positive Rate (FPR)**: Of those actually negative, how many were incorrectly predicted positive? $FPR = \frac{FP}{FP+TN}$. Used in ROC curves.
- **ROC Curve (Receiver Operating Characteristic)**: Plots TPR (Recall) vs. FPR at various classification thresholds. A curve hugging the top-left corner indicates better performance.

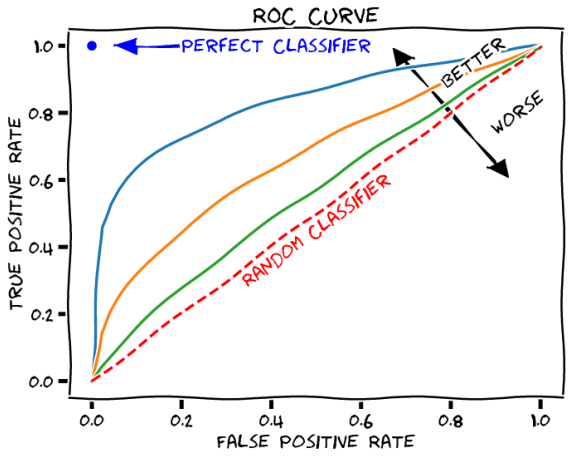


Figure 13: ROC Curve

- **AUC (Area Under the ROC Curve)**: Summarizes ROC performance into a single number (0 to 1). AUC=1 is perfect, AUC=0.5 is random guessing. Preferred metric for imbalanced datasets.
- **Multi-class Evaluation**:
  - **Macro-averaging**: Compute metric (P, R, F1) for each class independently, then average the results. Treats all classes equally.
  - **Micro-averaging**: Aggregate the counts (TP, FP, FN) across all classes, then compute the metric once. Gives more weight to larger classes.

10.10 Development/Test Sets and Cross-Validation

Essential practices for reliable model evaluation and development.

- **Data Splitting**:
  - **Training Set**: Used to train the model parameters (e.g., estimate NB probabilities, find LR weights).
  - **Development Set (Dev Set / Validation Set)**: Used to tune hyperparameters (e.g., smoothing value in NB, regularization strength in LR, network architecture), select features, and make modeling decisions. NOT used for final evaluation.
  - **Test Set**: Used ONLY ONCE for the final evaluation of the chosen model's performance. Must be completely unseen during development. Prevents "tuning to the test set".
  - *(Optional) Dev-Test Set*: Sometimes used for preliminary testing before the final test set.
- **Cross-Validation (k-fold CV)**:
  - Used when data is limited or to get a more robust performance estimate.
  - Split the data into k folds.
  - Train on k-1 folds, evaluate on the held-out fold.
  - Repeat k times, holding out a different fold each time.
  - Average the results across the k folds.
  - Helps mitigate sampling errors from a single train/dev split.

10.11 Overfitting and Complex Features

Overfitting occurs when a model learns patterns that are specific to the training data, including noise or accidental correlations, rather than generalizable patterns.

Complex features increase the risk of overfitting because:

- They may capture irrelevant or rare patterns that do not appear in the test data.
- The model might assign high importance to such features, leading to poor generalization.

Example: A rare phrase in the training set may cause the model to predict a specific class with high confidence, but this pattern may never appear again in real use.

Conclusion: Simpler features often generalize better. Always validate on unseen data to detect overfitting.

11 Word Semantics and Word Vectors

11.1 Lexical Semantics Basics

Linguistic study of word meaning:

- **Lemma**: The canonical or dictionary form of a word (e.g., "be" is the lemma for "is", "are", "am").
- **Word Sense**: A specific meaning of a word.
- **Polysemy**: A word with **multiple related meanings** (e.g., "bank" - financial institution vs. building, but not river's bank(*Homonymy*)). Most common words are polysemous.
- **Homonymy**: Words that **share the same form** (spelling/pronunciation) but have unrelated meanings (e.g., "bat" - animal vs. equipment; "bank" - institution vs. river side).
  - ***Homographs***: **Same spelling**, different meanings (may have different pronunciation, e.g., "bass" fish vs. instrument).
  - ***Homophones***: **Same pronunciation**, different meanings (e.g., "write"/"right", "piece"/"peace").
- **Word Sense Disambiguation (WSD)**: Task of identifying the correct sense of a word in its context. (machine translation, question answering, sentiment analysis, text-to-speech*(homographs like bass and bow)*) **Two Variants of WSD Task:**

  - **Lexical Sample Task:**
    * Focuses on a small, **pre-selected set of target words** (e.g., *line, plant, bass*).
    * Uses a known list of senses for each word.
    * A classifier is **trained separately for each word** using supervised learning.
  - **All-Words Task:**
    * Tries to **disambiguate every word in a full text**.
    * Uses a dictionary (lexicon) with all possible senses.
    * Harder to train specific classifiers because of data sparsity.

- **Semantic Similarity/Relatedness Calculation**: Task of quantifying the semantic closeness or relatedness between words or concepts, often leveraging semantic hierarchies or distributional information.
- **Named Entity Disambiguation (NED) / Fine-grained Named Entity Recognition (NER)**: Task of identifying the specific real-world entity a name refers to (NED) or classifying entities into highly specific semantic types (Fine-grained NER), essentially WSD applied to proper nouns or detailed entity classifications.
- **Lexical Distance**: A quantitative measure of semantic dissimilarity between two words, often defined as the length of the shortest path between them in a semantic hierarchy (ontology or lexical resource). Lower distance implies higher semantic similarity.
- **Lexical Cohesion**: A measure of the semantic connectedness or relatedness within a text segment. It is typically calculated as the average lexical distance between all unique pairs of content words within that segment. Lower average distance indicates higher cohesion.

$$\text{Lexical Cohesion} = \frac{\text{Sum of lexical distances between all pairs of words}}{\text{Number of pairs}}$$

- **Semantic Relations**:
  - **Synonymy**: Words with (nearly) the same meaning (e.g., couch/sofa). Perfect synonymy is rare *(a word that completely replaces another in all situations, without the slightest difference)*; often differ in connotation, register, politeness.
  - ***Linguistic Principle of Contrast*** - Difference in form → difference in meaning
  - **Antonymy**: Words with opposite meanings, differing **along one dimension** (e.g., hot/cold, long/short, rise/fall). Can: define a binary opposition or be at opposite **ends of a scale** (*long/short*, *fast/slow*), be **reversives** (*rise/fall*, *up/down*)
  - **Similarity**: Not synonyms, but sharing some element of meaning (e.g., car/bicycle (both are transport), cow/horse (animals)). Graded concept.
  - **Relatedness (Association)**: Words related in any way, often through a semantic field (e.g., car/gasoline - related but not similar; scalpel/surgeon - related within hospital field, pizza/pasta - both Italian food(are not similar in terms of ingredients, preparation or appearance. So, they are related but not similar.)).
  - **Semantic field**: A group of words that belong to the same **semantic domain and are related in meaning**, such as *surgeon, scalpel, nurse* (hospitals), *waiter, menu, chef* (restaurants), or *door, kitchen, bed* (houses).
  - **Hypernymy/Hyponymy (IS-A relation)**:
    * ***Hypernym (Superordinate)***: broader concept (e.g., "vehicle" is a hypernym of "car").
    * ***Hyponym (Subordinate)***: more specific concept (e.g., "car" is a hyponym of "vehicle"; "mango" is a hyponym of "fruit").

Superordinate	**vehicle**	**fruit**	**furniture**
Subordinate	car	mango	chair

Table 10: Examples of superordinate-subordinate (hypernym-hyponym) relations

  - **Meronymy/Holonymy (PART-OF relation)**: e.g. wheel/car.

- **Connotation**: Emotional meaning or evaluation associated with a word (e.g., positive/negative, happy/sad). **SentiWordNet** provides scores for this. (*is an automatically generated dictionary that assigns three scores to each sense of a word (from WordNet):: positivity, negativity, objectivity)*

11.2 Lexical Resources

- **WordNet**: A large lexical database of English. Nouns, verbs, adjectives, and adverbs are grouped into sets of cognitive synonyms (**synsets**), each expressing a distinct concept. Synsets are interlinked by means of conceptual-semantic and lexical relations (hypernymy, hyponymy, meronymy etc.). Forms a hierarchy/graph.

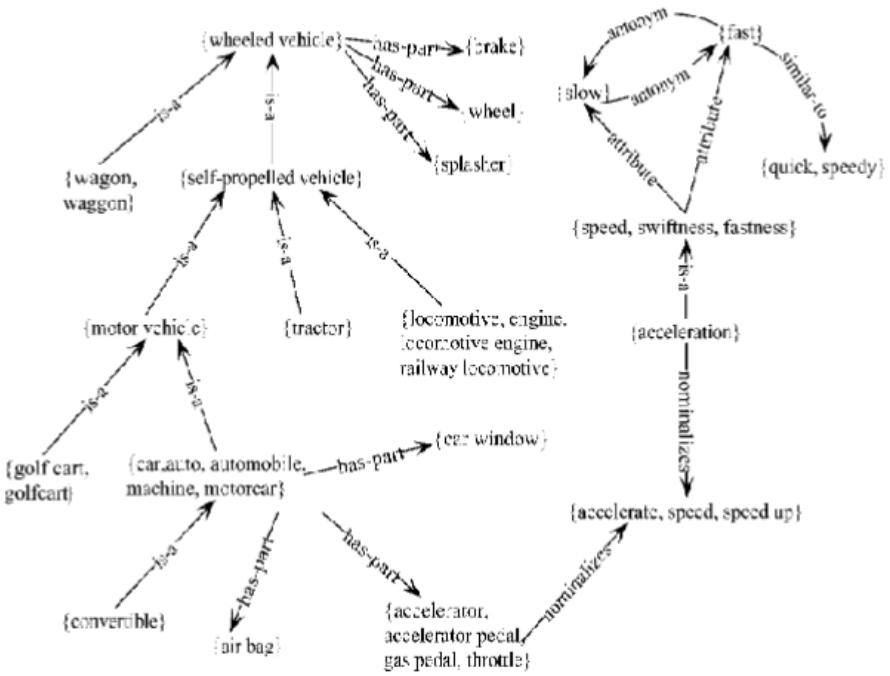


Figure 14: WordNet: Viewed as a graph

Python Example (NLTK WordNet):

```
import nltk
# nltk.download('wordnet') # Run once
# nltk.download('omw-1.4') # Optional: Open Multilingual Wordnet
from nltk.corpus import wordnet as wn

# Get synsets for a word
syns = wn.synsets('dog')
print("Synsets for 'dog':", syns)
# Output: [Synset('dog.n.01'), Synset('frump.n.01'), ..., Synset('chase.v.01')]

# Get info about a specific synset
dog_n1 = wn.synset('dog.n.01')
print("\nDefinition:", dog_n1.definition())
# Output: Definition: a member of the genus Canis...
print("Examples:", dog_n1.examples())
# Output: Examples: ['the dog barked all night']
print("Lemmas:", dog_n1.lemmas())
# Output: Lemmas: [Lemma('dog.n.01.dog'), Lemma('dog.n.01.domestic_dog'), ...]
```

```
# Relations
print("Hypernyms:", dog_n1.hypernyms())
# Output: Hypernyms: [Synset('canine.n.02'), Synset('domestic_animal.n.01')]
print("Hyponyms:", dog_n1.hyponyms())
# Output: Hyponyms: [Synset('basenji.n.01'), Synset('corgi.n.01'), ...]

# Similarity between synsets (Path Similarity)
cat_n1 = wn.synset('cat.n.01')
print("\nPath similarity dog/cat:", dog_n1.path_similarity(cat_n1))
# Output: Path similarity dog/cat: 0.2
```

11.3 Vector Semantics: Dense Word Embeddings

Representing words as dense, low-dimensional, real-valued vectors, capturing semantic meaning.

- **Motivation**: Overcome limitations of sparse vectors (e.g., one-hot) which are high-dimensional, orthogonal, and lack similarity information.
- **Goal**: Create embeddings where vector similarity (e.g., cosine similarity) corresponds to word meaning similarity/relatedness. Allows generalization to unseen words.
- **Dimensionality**: Typically 50-1000 dimensions, much smaller than vocabulary size $|V|$.
- **Distributional Hypothesis**: Words that **occur in similar contexts tend to have similar meanings**. This is the foundation for learning word embeddings from text corpora.

11.4 Learning Word Vectors

- **BOW-based (Bag of Words):**
  - Old, simple method for representing words.
  - Words are turned into vectors based on how often they appear.
  - Vectors are *sparse* (mostly zeros).
  - Ignores context and word meaning.
  - *No learning involved* — just counting word frequencies.
- **Neural Embeddings (e.g., word2vec):**
  - Uses machine learning to create word vectors.
  - Vectors are *dense* (compact and informative).
  - Based on predicting nearby words.
  - *Trained using tasks like Skip-gram or CBOW, where the model learns to predict surrounding words or fill in missing words.*
- **Contextual Neural Embeddings (e.g., BERT):**
  - Vectors depend on the whole sentence (context).
  - Also *dense*, but more accurate.
  - Word meaning changes depending on surrounding words.
  - *Trained using masked language modeling (MLM) — the model learns to predict missing words in a sentence using full context.*

11.5 Word2Vec (Mikolov et al., 2013)

A popular framework for learning word embeddings using neural networks. It's a **predictive** (learning vectors to improve the predictive ability) model. ***Looks at "do words occur next to each other?" (yes/no)***

- **Core Idea**: Instead of counting co-occurrences, train a model to predict words based on their context, or context based on words. The learned weights become the word embeddings.
- **Two Main Architectures**:

- **Continuous Bag-of-Words (CBOW)**: Predicts the current (target) word based on the sum or average of the vectors of the surrounding context words (within a window).
- **Skip-gram (SG)**: Predicts the surrounding context words (within a window) given the current (target) word. Generally performs better, especially for infrequent words.

11.5.1 Skip-gram Training with Negative Sampling (SGNS)

Efficient training method.

- *Problem*: Softmax over the entire vocabulary (to predict context words) is computationally very expensive.
- *Solution*: Reformulate as a binary classification task. For each target word w and a true context word c_{pos} (positive example), generate k negative samples c_{neg} (words not in the context, sampled typically based on unigram frequency).
- *Objective*: Train a model (conceptually, logistic regression units) to distinguish positive pairs from negative pairs. Maximize the probability of positive pairs and minimize the probability of negative pairs.
- **Negative Sampling** - Instead of having a huge output space where each word is a separate class, use *positive* and *negative* examples.
  - This turns the problem into binary classification instead of multiclass classification.

  The challenge is how to create good positive and negative pairs — i.e., pairs of words that appear together (or not).
  - This is known as **contrastive learning**.
- *Loss Function (simplified, for one positive and k negative samples)*: Minimize the negative log-likelihood:

$$L = -\log \sigma((v'_{c_{pos}})^T v_w) - \sum_{i=1}^{k} \log \sigma(-(v'_{c_{neg_i}})^T v_w)$$

  where v_w is the "input" embedding for target word w (from matrix W), and v'_c is the "output" embedding for context word c (from matrix W' or C). σ is the sigmoid function.
- *Learning*: Use Stochastic Gradient Descent (SGD) to update the embedding vectors v_w and v'_c to minimize the loss.
- *Result*: Two sets of embeddings (W and W'). Usually, the input embeddings W are kept.

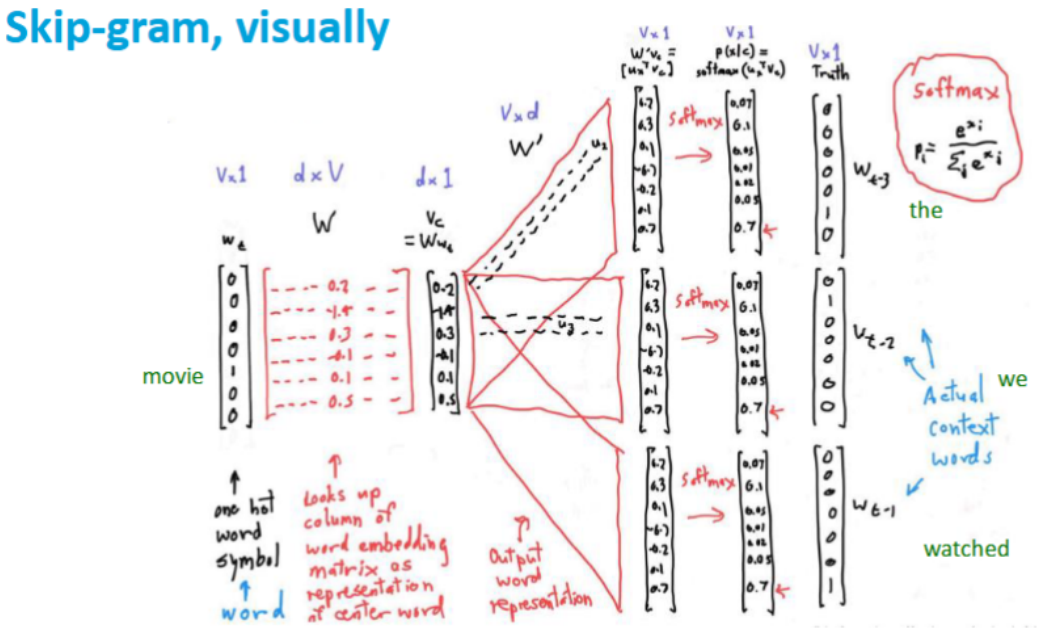


Figure 15: Skip-gram, visually

11.5.2 Word2Vec Learning

Word2Vec learns **two vectors for each word:**

- **Input vector** – when the word is the center (target word, in the W matrix)
- **Output vector** – when the word is in the context (neighbor word, in the W')

For the **final word vector** for a word we need to consider either the input vector v (from W matrix) or the output vector u (from W' matrix) corresponding to that word **or combine them in some way:**

- **Sum or average** the input and output vectors

Example:

$$\text{final\_vector(word)} = \frac{1}{2}(\text{input} + \text{output})$$

Limitations and Semantic Artifacts : Semantic hallucinations. For example, it might correctly infer analogies like "king - man + woman = queen," but it can also generate false relationships, especially when trained on limited or biased data. Additionally, Word2Vec **does not distinguish between different meanings of the same word**, as it generates only **one vector representation per word**.

If words appear frequently in similar contexts, the model starts to give them similar vectors. So words like king and queen or doctor and nurse will get similar vectors - not because they are the same, but because they **appear in similar situations.**

11.6 GloVe (Global Vectors) (Pennington et al., 2014)

An alternative approach based on global matrix factorization. It's a **count-based** model. ***"How many times do words meet next to each other"***

- **Core Idea**: Learn vectors such that their dot product relates directly to the logarithm of their co-occurrence count in the corpus.
- **Input**: Global co-occurrence matrix X, where X_{ij} is the number of times word j appears in the context of word i.
- **Objective (simplified)**: Minimize the difference between the dot product of word vectors and the log co-occurrence count:

$$J = \sum_{i,j=1}^{|V|} f(X_{ij})(w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij})^2$$

  where w_i and $\tilde{w}_j$ are the word and context vectors, $b_i, \tilde{b}_j$ are biases, and $f(X_{ij})$ is a weighting function to down-weight very frequent co-occurrences.

- Captures **global statistics directly**, unlike the local context windows of Word2Vec. Because:
  - It builds a large table of how often each word appears near other words.
  - For example, if *"coffee"* often appears near *"cup"*, they are considered related.
  - This co-occurrence information is then used to learn word vectors.

11.7 FastText (Bojanowski et al., 2017)

An **extension of Word2Vec** (usually Skip-gram) that incorporates subword information.

- **Core Idea**: Represent **each word as a bag of character n-grams** (typically 3 to 6 characters), in addition to the word itself.
- Example: For `"where"` with $n = 3$: <`wh`, `whe`, `her`, `ere`, `re`>. (< and > mark boundaries).
- The **vector for a word is the sum of the vectors corresponding to its character n-grams.**
- **Advantages**:

  - Can generate embeddings for OOV words based on their n-grams.
  - More robust to typos.
  - Captures morphological similarities (e.g., "running", "runner" will share n-grams like "run").
- Similar training process to Word2Vec (e.g., SGNS).

11.8 Summary: How to Learn word2vec Embeddings

- Start with $|V|$ random d-dimensional vectors as initial embeddings.
- Use logistic regression (or any classifier):
  - Take a corpus and select word pairs that **co-occur** as **positive examples**.
  - Take word pairs that **do not co-occur** as **negative examples**.
  - Train a classifier to tell them apart by adjusting the word vectors.
  - After training, discard the classifier and keep the learned word vectors (matrices W and W').

11.9 Evaluating Word Embeddings

- **Intrinsic Evaluation**: Evaluate embeddings based on their **ability to capture semantic relationships directly.**
  - ***Word Similarity*:**
    - **Task**: Compute cosine similarity **between word vectors and compare with human similarity** ratings on benchmark datasets (e.g., **WordSim353** - focuses on **relatedness** (When estimating similarity of **antonyms**, consider them "similar" (i.e., **belonging to the same domain** or representing features of the same concept), rather than "dissimilar".), **SimLex-999** - focuses on **paraphrastic similarity**) (two words are "similar" if they have similar meanings).
    - **Metric**: Spearman's rank correlation coefficient (ρ) between model similarities and **human ratings.**

Table 11: WordSim353
(Finkelstein et al., 2002) (0 = words are unrelated)

word pair		similarity
journey	voyage	9.3
king	queen	8.6
computer	software	8.5
law	lawyer	8.4
forest	graveyard	1.9
rooster	voyage	0.6

Table 12: SimLex-999
(Hill et al., 2014)

word pair		similarity
insane	crazy	9.6
attorney	lawyer	9.4
author	creator	8.0
diet	apple	1.2
new	ancient	0.2

  - ***Word Analogy*:**
    - **Task**: Test relationships like "a is to b as c is to ?" (e.g., "king" - "man" + "woman" = ?). Find word d whose vector is closest to $v_b - v_a + v_c$.
    - **Metric**: Accuracy on analogy questions (e.g., Google analogy dataset, SemEval tasks).
- **Extrinsic Evaluation**: Evaluate embeddings based on their contribution to performance on a **downstream NLP task** (any practical natural language processing task that uses ready-made textual representations...) and judge the performance.
  - **Task**: Use the pre-trained embeddings as input features for a task-specific model (e.g., text classification, NER).
  - **Metric**: Performance metric of the downstream task (e.g., F1 score, accuracy).
  - Considered more definitive but can be harder to interpret the specific contribution of the embeddings.

11.10 Properties and Issues

11.10.1 Properties

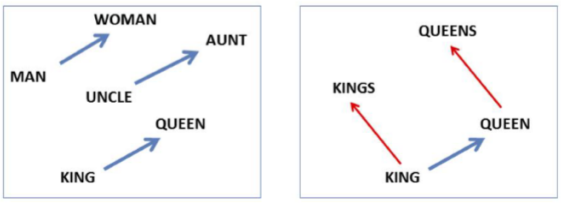


Figure 16: Embeddings (should) capture relational meaning

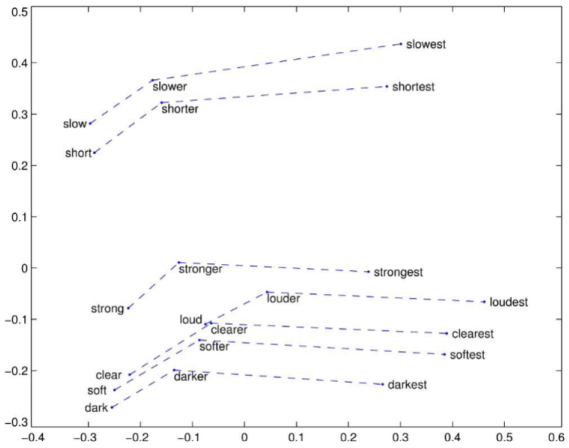


Figure 17: Embeddings capture morphological and semantic regularities

- **Context Window Size (hyperparameter)**: Affects the type of similarity captured. **Smaller windows (+/- 2)** tend to capture more **syntactic/taxonomic similarity**, while **larger windows (+/- 5)** capture more **topical/semantic relatedness**.
- In **small corpora, word2Vec *tends to* outperform GloVe** (because Word2Vec is trained on the local context of words within a sliding window, which can be more effective in capturing the nuances of a smaller dataset)
- **Isotropy**: Good embeddings are evenly spread in space. If they are too clustered, the model may struggle to tell words apart.
- **Compositionality**: Embeddings can be combined to represent **multi-word expressions**.
  Example: `vector("small house")` ≈ `vector("small")` + `vector("house")`
- **Analogy Solving**: Embeddings capture **logical relationships between words**.
  Example: `king - man + woman` ≈ `queen`

Equation	Valid?	Explanation
`boy - girl = brother - sister`	Yes	Captures **gender difference between similar roles**.
`boy - brother = girl - sister`	Yes	Captures **shift from child to sibling within the same gender**.
`boy - brother = sister - girl`	No	The analogy is not symmetric; **direction and logic mismatch**.
`sister - brother = boy - girl`	No	The **relation direction is inconsistent**; may reverse meaning.

Table 13: Evaluating analogy equations in word embeddings

- **Domain Sensitivity**: Embeddings trained on different types of data (e.g., tweets vs. news) can behave differently. **It's important to match the training data to the task**.

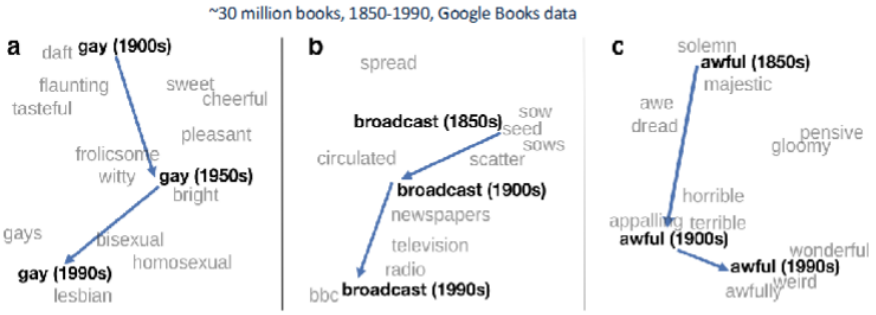


Figure 18: Train embeddings on different decades of historical text to see meanings shift

11.10.2 Issues

- **Bias**: Embeddings trained on large, real-world text data **inherit societal biases present in the text** (e.g., gender stereotypes like `man:doctor :: woman:nurse`). Debiasing embeddings is an active research area.
- **Static Nature**: Traditional embeddings (Word2Vec, GloVe, FastText) **assign a single vector to each word type, regardless of context** (cannot distinguish polysemy well). Contextual embeddings address this.
- **Out-of-Vocabulary (OOV)**: Pretrained embeddings cannot handle words that were not seen during training (e.g., typos, new slang, rare names). Some models like **FastText mitigate this using subword information.**
- **Domain Mismatch**: Embeddings trained on one type of data (e.g., news) may perform poorly on different domains (e.g., biomedical or social media). Domain-specific retraining or adaptation is often required.
- **Lack of Interpretability**: Word embeddings are dense vectors and hard to interpret. **It is difficult to explain what each dimension represents or why two vectors are similar.**
- **Storage and Efficiency**: Large embedding matrices can consume significant memory, especially in multilingual or very large-vocabulary settings.

Python Example (Using Gensim to load pre-trained models):

```
# Need to install gensim: pip install gensim
# Download a pre-trained model, e.g., GloVe or Word2Vec
# Example using GloVe (e.g., download glove.6B.100d.txt from Stanford website)
# Need a utility to convert GloVe text format to Word2Vec binary format
# from gensim.scripts.glove2word2vec import glove2word2vec
# glove_input_file = 'glove.6B.100d.txt'
# word2vec_output_file = 'glove.6B.100d.word2vec'
# glove2word2vec(glove_input_file, word2vec_output_file)

from gensim.models import KeyedVectors

# Load the converted model
# model_path = 'glove.6B.100d.word2vec' # Or use a Word2Vec binary file
# model = KeyedVectors.load_word2vec_format(model_path, binary=False) # Set binary=True for Word2Vec

# Alternatively, use gensim's downloader (easier!)
import gensim.downloader as api
model_name = 'glove-wiki-gigaword-100' # Example model
print(f"Loading model: {model_name}...")
model = api.load(model_name)
print("Model loaded.")
```

```
# Get vector for a word
try:
    vector_king = model['king']
    print(f"\nVector for 'king' (first 5 dims): {vector_king[:5]}")
    print(f"Vector dimension: {len(vector_king)}")

    # Find most similar words
    similar_to_king = model.most_similar('king', topn=5)
    print("\nMost similar to 'king':", similar_to_king)
    # Output might be: [('queen', 0.7), ('prince', 0.6), ...]

    # Perform analogy task: king - man + woman = ?
    analogy_result = model.most_similar(positive=['woman', 'king'], negative=['man'], topn=1)
    print("\nAnalogy 'king - man + woman':", analogy_result)
    # Output might be: [('queen', 0.7...)]

    # Calculate similarity
    similarity = model.similarity('king', 'queen')
    print(f"\nSimilarity between 'king' and 'queen': {similarity:.4f}")

except KeyError as e:
    print(f"\nWord not in vocabulary: {e}")
```

12 Neural Network Models for NLP

12.1 Using Word Embeddings in Neural Networks Dense word embeddings (like Word2Vec, GloVe, FastText) serve as input features to neural networks.

- **Strategies for using embeddings**:
  1. **Train from scratch**: Initialize embeddings randomly and learn them as part of the network training. Requires large task-specific dataset.
  2. **Use pre-trained (frozen)**: Load embeddings trained on a large corpus (e.g., Google News Word2Vec, GloVe Common Crawl). Keep these embeddings fixed ("frozen") and only train the subsequent layers of the task-specific network. Good when task data is limited.
  3. **Use pre-trained (fine-tuning)**: Initialize with pre-trained embeddings but allow them to be updated (fine-tuned) during task-specific training. Often gives the best performance if sufficient task data is available.
- **Different sizes of inputs:**
  - **Assuming fixed-length inputs:**
    - Use simple models like Deep Averaging Networks or Feedforward Neural Networks (FFNN)
    - Suitable for text classification or basic language models
  - **Assuming variable-length inputs:**
    - Use sequence models that process word order:
      - RNNs (Recurrent Neural Networks)
      - CNNs (Convolutional Neural Networks)
      - Transformers (modern architectures)
    - Suitable for tasks like text generation, translation, dialogue systems
- **Aggregating Embeddings for Fixed-Size Input**: To handle variable-length sequences for models requiring fixed-size input (like basic FFNNs).

– **Average Word Embeddings (NBOW - Neural Bag of Words / DAN - Deep Averaging Networks)**: Compute the element-wise average of the embeddings of all words in the sequence.

$$h(x) = \frac{1}{n}\sum_{i=1}^{n} \text{emb}(x_i)$$

Simple but effective baseline for text classification, ignores word order. We can treat h(x) as the whole sentence embedding.

– **Sum Word Embeddings**: Compute the element-wise sum.

$$h(x) = \sum_{i=1}^{n} \text{emb}(x_i)$$

12.2 Neural Language Models (LMs) Definition

- **Language Modeling:** Calculating the probability of the next word in a sequence given some history.
  - We've seen N-gram based LMs
  - But neural network LMs far outperform n-gram language models
- State-of-the-art neural LMs are based on more powerful neural network technology (like Transformers)
- But **simple feedforward LMs** can do almost as well!

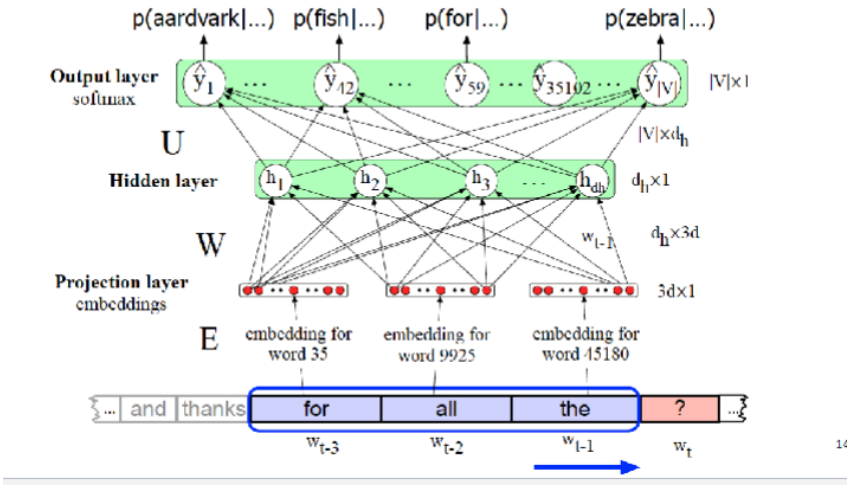


Figure 19: Neural Language Model: FFNN

12.3 Feedforward Neural Language Models (FFNN-LM)

- **Task**: Predict the next word w_t given a fixed-size window of preceding words $w_{t-N+1}, ..., w_{t-1}$. Approximates $P(w_t|w_1^{t-1}) \approx P(w_t|w_{t-N+1}^{t-1})$.
- **Architecture**:
  1. Retrieve embeddings for the $N-1$ context words.
  2. Concatenate these embeddings into a single input vector.
  3. Pass through one or more hidden layers (e.g., with ReLU or tanh activation).
  4. Output layer with Softmax activation over the entire vocabulary $|V|$ to produce probabilities for the next word.

$$\hat{y}_t = \text{softmax}(Vh + b_y)$$

where $h = g(W[\text{emb}(w_{t-N+1}); ...; \text{emb}(w_{t-1})] + b_h)$.

- **Advantage over N-grams**: Can **generalize using semantic similarity of embeddings** (e.g., predict "fed" after "dog" if seen "fed" after "cat").
- **Limitation**: Fixed context window size; **cannot model long-range dependencies beyond the window.**

12.4 Convolutional Neural Networks (CNNs) for NLP

Apply convolutions (filters) over sequences of word embeddings to capture local patterns (like n-grams).

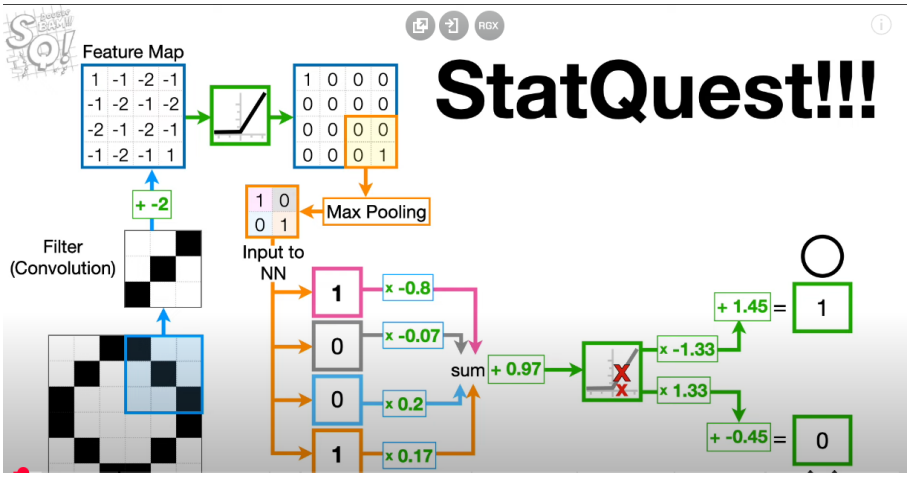


Figure 20: Convolutional Neural Network for image classification. StatQuest

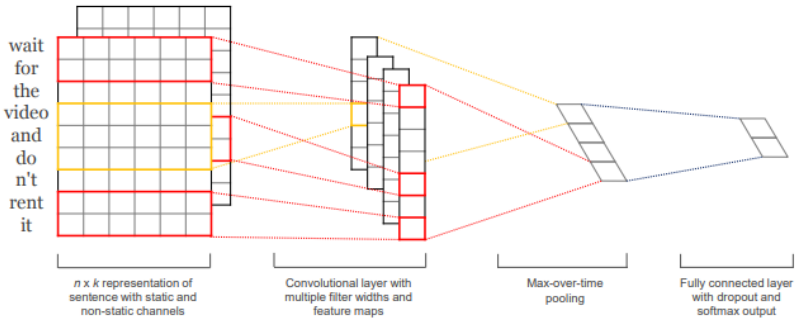


Figure 21: CNN architecture for the text analysis

- **Filters**: Small matrices (e.g., size $m \times k$, where m is window size/n-gram size, k is embedding dimension) that slide over the input sequence. Each filter detects a specific pattern.
- **Convolution Operation**: Applying a filter to a window of m words involves computing element-wise products and summing them up (similar to dot product). Sliding the filter produces a *feature map*.
- **Multiple Filters**: Use multiple filters of potentially different window sizes (m) to capture various patterns (e.g., unigram, bigram, trigram features).
- **Pooling Layer (Max-pooling / Average-pooling)**: Reduces the dimensionality of the feature maps while retaining important information.
  - *Max-pooling over time*: Takes the maximum value from each feature map. Captures the most salient feature detected by the filter, regardless of its position. Results in a fixed-size vector.
  - *Average-pooling*: Takes the average value from each feature map.

- **Typical Architecture for Text Classification**:
  1. Input: Matrix of word embeddings ($n \times k$, where n is sequence length).
  2. Convolutional Layer(s): Apply multiple filters of different sizes (e.g., 2, 3, 4-grams).
  3. Activation Function (e.g., ReLU).
  4. Max-pooling Layer: Apply max-pooling over the time dimension for each feature map.
  5. Concatenate pooled features from all filters into a single fixed-size vector.
  6. Fully Connected Layer(s) + Softmax Output: For classification.
- **Advantages**: Captures local dependencies, relatively efficient, can be used for character-level processing. Can be used for the **language modeling** (use causally masked convolutions)
- **Limitations**: Fixed filter size limits the range of dependencies captured directly in one layer (though stacking layers helps).

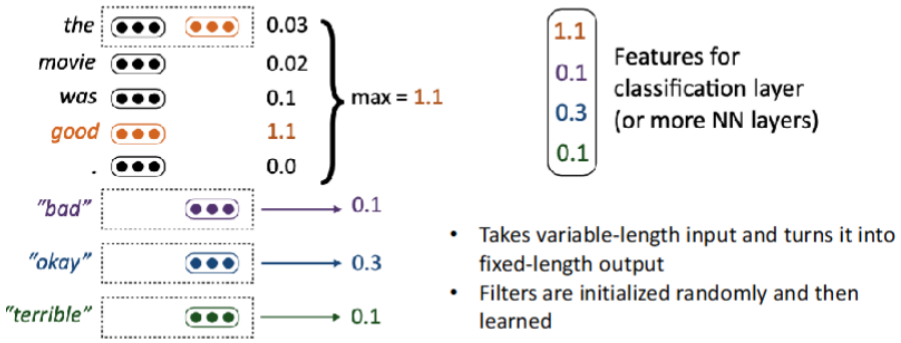


Figure 22: Example of the CNN for sentiment analysis

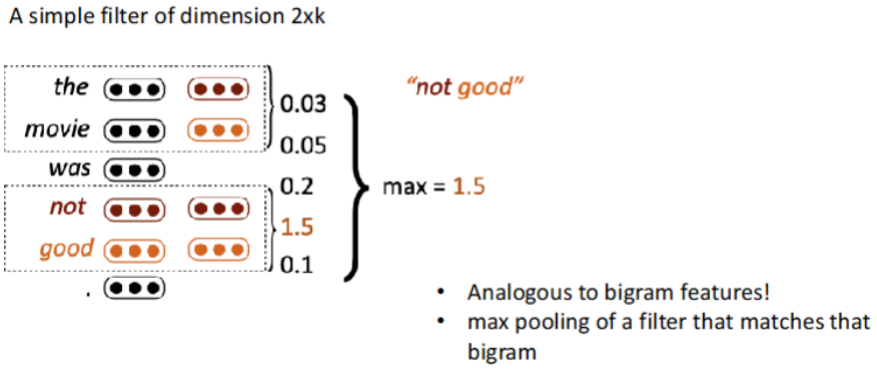


Figure 23: CNN architecture for the text analysis using filters of dimension 2xk

12.4.1 What is a Filter Map in CNNs for Text?

When applying a convolutional filter over a sentence (represented as word embeddings), the output is a series of activation values — this is called a **filter map** (or feature map).

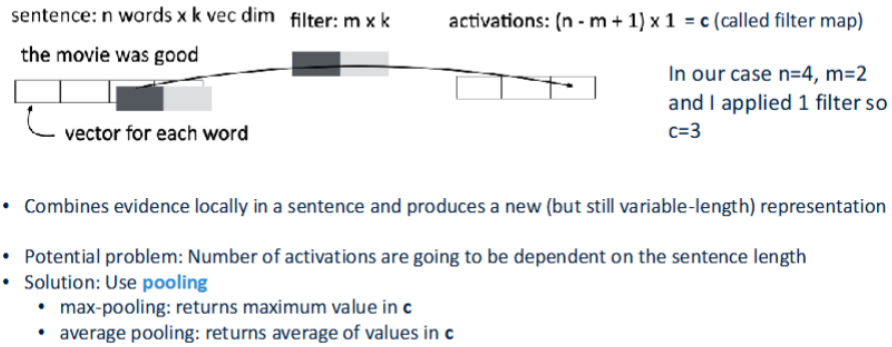


Figure 24: Example of the feature maps

- Suppose we have n words in a sentence, each with embedding of size k
- A filter of size $m \times k$ slides over the sentence and produces $(n - m + 1)$ values. m is the width of the word count filter (i.e. how many words the filter captures in one step, e.g. m=2=bigram).
- This sequence of values is the **filter map**

Example: For $n = 4$, $m = 2$ and 1 filter applied, the output is a filter map of size 3

$$\text{Filter Map Size} = n - m + 1 = 4 - 2 + 1 = 3$$

This captures local patterns (like phrases) in the input sequence.

12.4.2 Properties and Issues

Properties

- **Convolutional Layer:** Applies multiple filters over the input sequence (e.g., word embeddings), allowing the model to learn local patterns.
- **Filters capture n-grams:** Filters of different widths capture bigrams, trigrams, etc. Common range: 1 to 5 tokens.
- **Position Invariance via Pooling:** Pooling (max or average) reduces variable-length outputs to fixed-size representations and helps detect patterns regardless of position.
- **Efficient and Parallelizable:** Convolution can be computed in parallel over the whole sequence, making CNNs faster than RNNs.
- **Can be used at character level:** CNNs are suitable for character-level tasks. Filters can operate over character n-grams to learn morphological features.
- **Stackable Layers:** Multiple convolutional layers can be stacked to detect increasingly abstract patterns (e.g., words → phrases → sentence-level features).

Issues

- **Limited Context Window:** Each filter sees only a fixed number of tokens. Long-range dependencies are harder to capture without deep stacking.
- **No True Sequence Modeling:** CNNs don't model word order globally like RNNs or Transformers. They rely on local patterns only.
- **Loss of Detail in Pooling:** Max/average pooling reduces dimensionality but may discard important fine-grained information.
- **Harder to Interpret:** Like other neural models, it's unclear what each filter or dimension actually captures.
- **Requires Careful Design:** Choice of filter size, number of filters, and pooling strategy significantly affects performance and needs tuning.

12.5 Recurrent Neural Networks (RNNs)

Designed to process sequential data by maintaining a hidden state that captures information from previous steps.

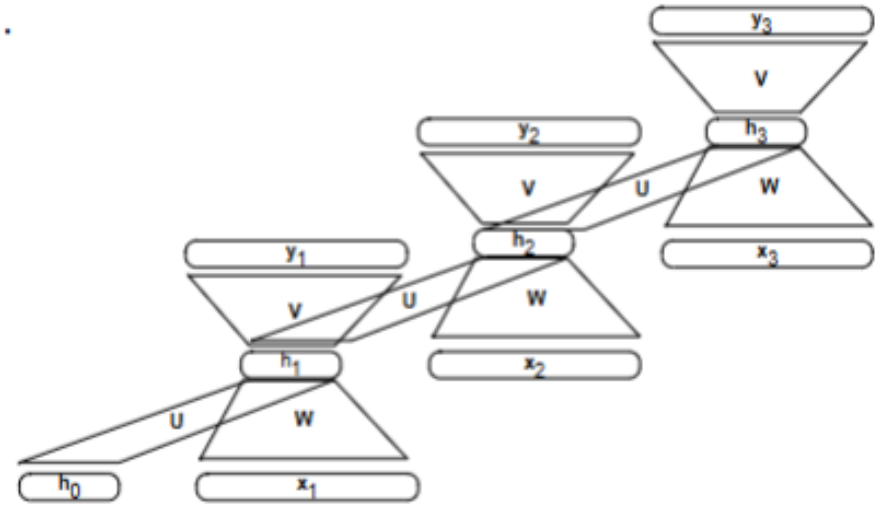


Figure 25: Unrolled Recurrent Neural Network (RNN). Each time step t processes: x_t – input (e.g., word embedding), h_t – hidden state summarizing previous inputs, y_t – output (e.g., prediction). Weight matrices: W – input to hidden, U – hidden to hidden (recurrent), V – hidden to output.

- **Core Idea**: The output (or hidden state) at time step t depends on the input at time t and the hidden state at time $t-1$. Allows information to persist.
- **Simple RNN (Elman Network) Equations**:

$$h_t = g(Wx_t + Uh_{t-1} + b_h)$$

$$y_t = f(Vh_t + b_y)$$

  where x_t is input vector at time t, h_t is hidden state, y_t is output. W, U, V are shared weight matrices. g, f are activation functions (e.g., $g = \tanh$, $f = \text{softmax}$).
- **Unrolling in Time**: For training/visualization, the RNN can be seen as a deep feedforward network where each layer corresponds to a time step and weights are shared across layers.
- **Training (Backpropagation Through Time - BPTT)**: Apply backpropagation to the unfolded network. Gradients are summed/averaged across time steps for shared weights W, U, V.
- **Vanishing/Exploding Gradients**: Major difficulty in training simple RNNs on long sequences. Gradients propagated over many steps can shrink to zero (vanish) or grow exponentially (explode), hindering learning of long-range dependencies. Exploding gradients are **often controlled with gradient element-wise or norm clipping**.

12.5.1 RNN Applications

- **Sequence Labeling**: Predict a label y_t for each input x_t (e.g., POS tagging, NER **(Named Entity Recognition)** Main metric for NER: Precision, Recall, F1-score). **Output is taken at each step.**

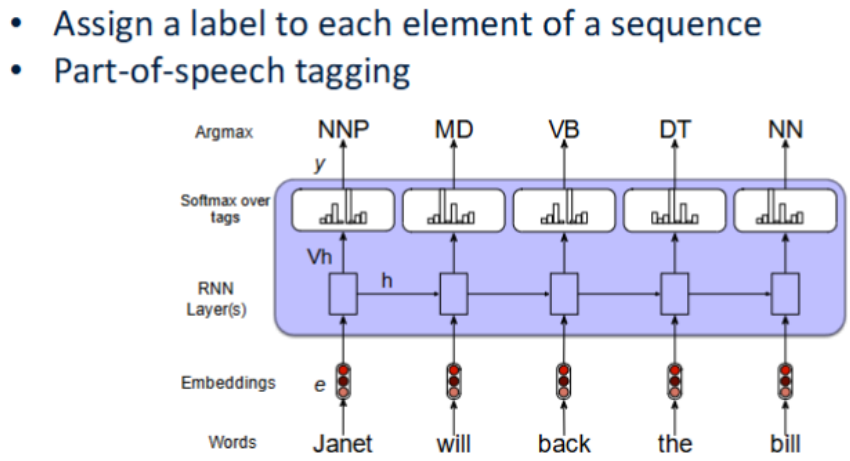


Figure 26: RNNs for sequence labeling

- **Sequence Classification**: Predict a single label for the entire sequence. Typically **uses the final hidden state h_n or an aggregation (e.g., mean/max pooling) of all hidden states** $h_1, ..., h_n$ as input to a classifier.

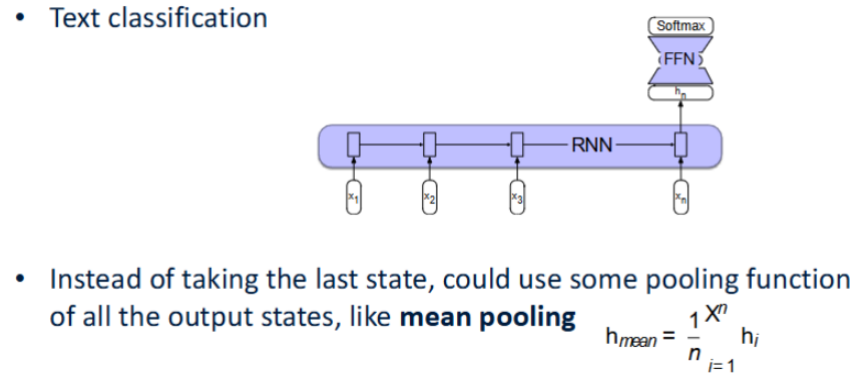


Figure 27: RNNs for sequence classification

- **Language Modeling / Sequence Generation**: Predict the next token $y_t = w_{t+1}$ based on the current state h_t. The predicted token (or its embedding) is then fed as input x_{t+1} for the next step (autoregressive).

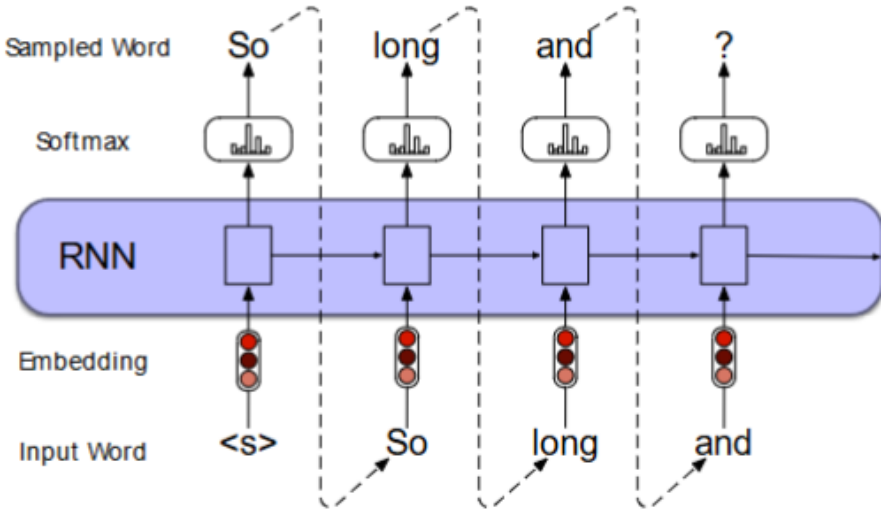


Figure 28: RNN language Modelling

12.5.2 Architectural Variants

- **Stacked RNNs**: Output hidden states of one RNN layer serve as input to the next RNN layer. Increases model capacity.
- **Bidirectional RNNs (BiRNNs)**: Process the sequence in both forward (h_t^f) and backward (h_t^b) directions using two separate RNNs. The final representation at time t is often the concatenation $[h_t^f; h_t^b]$. Captures context from both past and future. Suitable for tasks where the entire input is available (classification (based on concatenation of both final outputs (h1, hn)), labeling), but not for standard autoregressive generation.

12.5.3 Properties and Issues

Properties

- **Sequential Processing:** RNNs process input tokens one at a time, maintaining a hidden state that summarizes previous context.
- **Context Awareness:** Hidden state h_t stores information from previous time steps, enabling learning of temporal and linguistic dependencies.
- **Parameter Sharing:** The same weights are reused across all time steps, which reduces the number of parameters and helps generalization.
- **Suitable for Variable-Length Input:** RNNs naturally handle input sequences of different lengths without the need for padding or truncation.
- **Can Be Extended:** Variants like LSTM and GRU are designed to handle long-term dependencies more effectively than vanilla RNNs.
- **Good at:** speech recognition, weather forecasting, sequential data.

Issues

- **Vanishing/Exploding Gradients:** Gradients can become too small or too large during back-propagation through time, making training unstable — especially for long sequences.
- **Slow Training:** RNNs are inherently sequential; computations cannot be fully parallelized, which slows down training compared to CNNs or Transformers.
- **Short-Term Memory:** Standard RNNs struggle to retain information across long sequences. LSTM/GRU improve this, but still have limitations.

- **Difficult to Interpret:** The hidden states are dense and difficult to analyze or debug directly.
- **Sensitive to Initialization and Tuning:** RNNs can be unstable and require careful choice of learning rate, sequence length, and gradient clipping.

12.6 Long Short-Term Memory (LSTM) (Hochreiter & Schmidhuber, 1997)

A type of RNN specifically designed to mitigate the vanishing gradient problem and capture long-range dependencies.

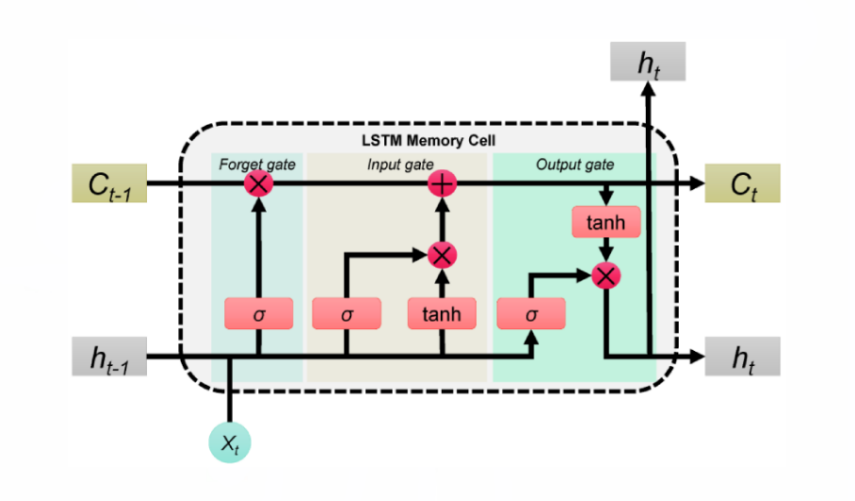


Figure 29: Long Short-Term Memory

- **Core Idea**: Introduce a *cell state* c_t that acts as a memory, and use *gates* to control the flow of information into and out of this memory.
- **Gates** (typically sigmoid activation, outputting values between 0 and 1):
  - **Forget Gate (f_t)**: Controls how much of the previous cell state c_{t-1} to forget.

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f)$$

  - **Input Gate (i_t)**: Controls how much of the new candidate information $\tilde{c}_t$ to add to the cell state.

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i)$$

$$\tilde{c}_t = \tanh(W_c x_t + U_c h_{t-1} + b_c) \quad \text{(Candidate cell state)}$$

  - **Output Gate (o_t)**: Controls how much of the cell state c_t to expose as the hidden state h_t.

$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o)$$

- **State Updates**:
  - **Cell State Update**: $c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$ ($\odot$ is element-wise multiplication).
  - **Hidden State Update**: $h_t = o_t \odot \tanh(c_t)$.
- The gating mechanism allows LSTMs to maintain information over long periods and selectively update memory.

12.6.1 Properties and Issues of LSTMs

Properties

- **Long-Term Dependency Handling:** LSTMs are designed to capture long-range dependencies by using gates to control the flow of information.
- **Memory Cell:** LSTMs maintain a separate memory cell c_t that stores information over time, enabling better retention than standard RNNs.
- **Gate Mechanisms:** Input, forget, and output gates help decide what to keep, forget, or output — giving LSTMs more control over information flow.
- **Sequential Processing:** Like RNNs, LSTMs process sequences step by step, maintaining temporal structure.
- **Effective for Sequential Tasks:** LSTMs perform well in tasks like language modeling, speech recognition, machine translation, and time series prediction.

Issues

- **Slow Training:** LSTMs are sequential and cannot be parallelized easily across time steps, which leads to slower training than CNNs or Transformers.
- **Complexity:** LSTMs are more complex than standard RNNs due to gating mechanisms, leading to more parameters and higher computational cost.
- **Still Sensitive to Long Ranges:** Although better than vanilla RNNs, LSTMs can still struggle with very long dependencies.
- **Hyperparameter Sensitivity:** Performance depends on careful tuning (e.g., hidden size, learning rate, dropout).
- **Interpretability:** Like other deep models, the inner workings of LSTMs (gates and states) are hard to interpret intuitively.

12.7 Gated Recurrent Unit (GRU) (Cho et al., 2014)

A simplification of the LSTM with fewer parameters.

- Combines the cell state and hidden state.
- Uses two gates:
  - **Update Gate (z_t)**: Controls how much of the previous state to keep vs. how much new candidate state $\tilde{h}_t$ to incorporate (combines LSTM's forget and input gates).
  - **Reset Gate (r_t)**: Controls how much the previous state h_{t-1} influences the candidate state $\tilde{h}_t$.
- **Operation at time t**: Let x_t be the input vector and h_{t-1} be the hidden state from the previous time step.

  1. **Calculate Gates:**
  $$r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r) \quad \text{(Reset Gate)}$$
  $$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z) \quad \text{(Update Gate)}$$
  2. **Compute Candidate Hidden State:**
  $$\tilde{h}_t = \tanh(W_h x_t + U_h(r_t \odot h_{t-1}) + b_h)$$
  3. **Compute Current Hidden State:**
  $$h_t = z_t \odot \tilde{h}_t + (1 - z_t) \odot h_{t-1}$$

  Where σ is the sigmoid function, tanh is the hyperbolic tangent function, $\odot$ denotes element-wise multiplication, and W, U, b are learned weight matrices and bias vectors.
- Often performs similarly to LSTM but is computationally slightly cheaper.

12.8 Encoder-Decoder Architecture (Sequence-to-Sequence / Seq2Seq)

A framework for mapping an input sequence of variable length to an output sequence of potentially different variable length.

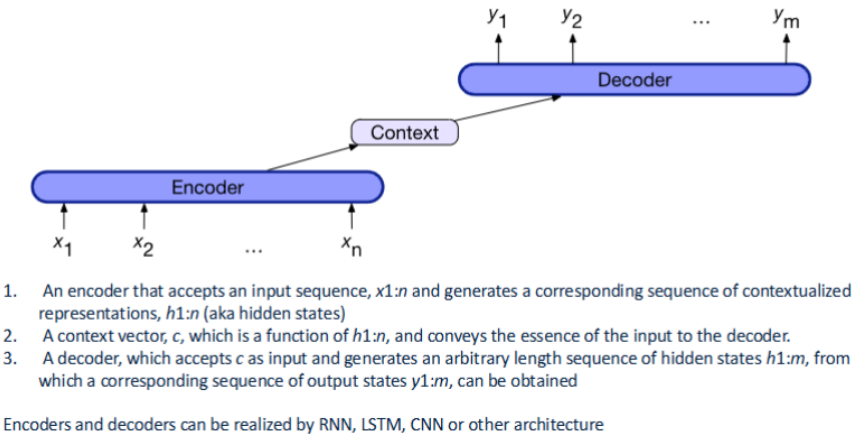


Figure 30: 3 components of an encoder-decoder

- **Components**:
  1. **Encoder**: An RNN (or CNN/Transformer) that processes the entire input sequence $x_1, ..., x_n$ and outputs a context representation c (often the final hidden state h_n^e).
  2. **Decoder**: An RNN that takes the context c and generates the output sequence $y_1, ..., y_m$ step-by-step. At each step t, it typically uses the context c, the previous hidden state h_{t-1}^d, and the previously generated output token $\hat{y}_{t-1}$ (or its embedding) to produce the next hidden state h_t^d and predict the next output token $\hat{y}_t$.
- **Equations (Basic RNN Decoder)**:

$$c = h_n^e$$
$$h_0^d = c \quad \text{(or initialized differently)}$$
$$h_t^d = g(\text{emb}(\hat{y}_{t-1}), h_{t-1}^d, c)$$
$$\hat{y}_t = \text{softmax}(V h_t^d)$$

$$h_t^d = g(\hat{y}_{t-1}, h_{t-1}^d, c)$$

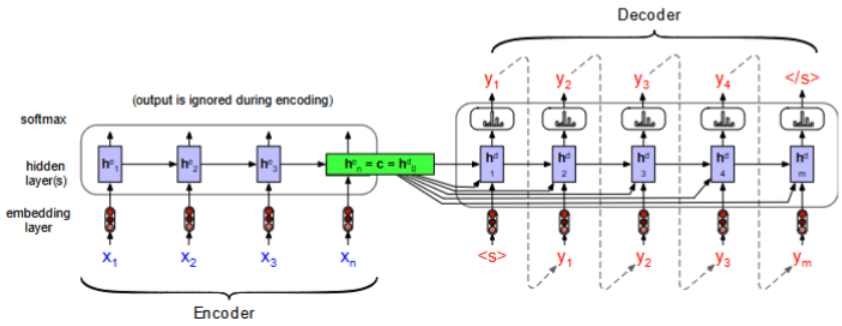


Figure 31: Encoder-decoder showing context. The context vector c_i computed at decoding step i as a weighted sum of all encoder hidden states, with weights determined by the similarity between the decoder's previous hidden state and each encoder hidden state

- **Teacher Forcing**: During training, instead of feeding the decoder's own previous prediction $\hat{y}_{t-1}$, feed the ground truth token y_{t-1}. Helps stabilize training but can cause mismatch between training and inference.

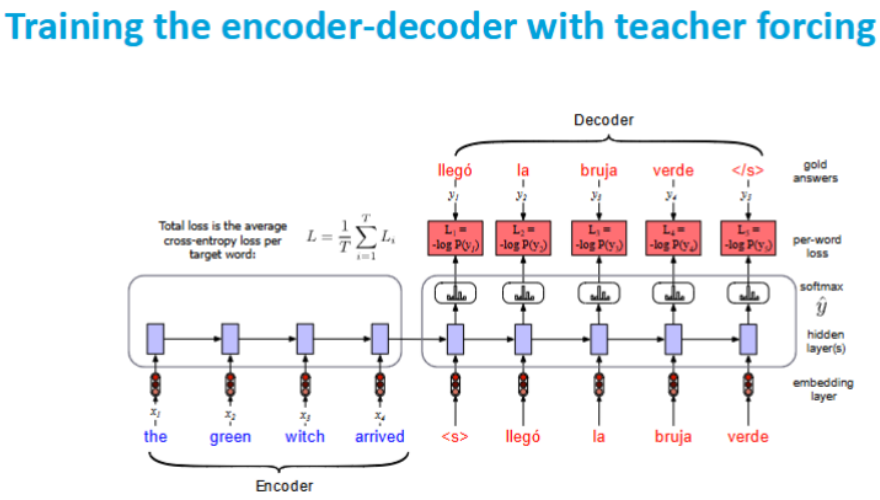


Figure 32: Training the encoder-decoder with teacher forcing

- **Limitation (Bottleneck Problem)**: Compressing the entire input sequence into a single fixed-size context vector c can lead to information loss, especially for long sequences. This motivated the development of Attention mechanisms.

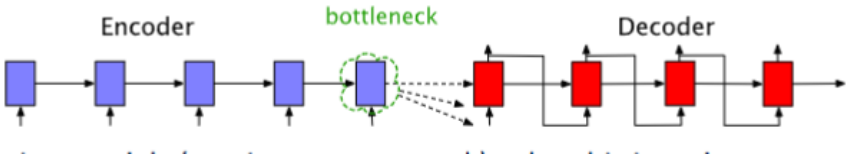


Figure 33: Limitations of simple encoder (+decoder) models

13 Attention and Transformers

13.1 Limitations of Basic Encoder-Decoder Models

- The fixed-size context vector c (typically the last hidden state of the encoder) becomes a **bottleneck**, making it hard to summarize **all information from long input sequences.**

13.2 Attention Mechanism (in Seq2Seq) Allows the decoder to dynamically focus on different parts of the input sequence at each decoding step. Solves the bottleneck problem.

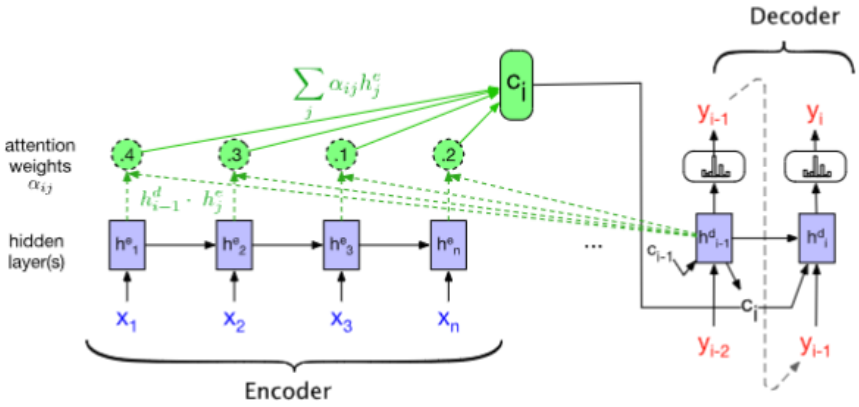


Figure 34: Encoder-decoder with attention, focusing on the computation of c

- **Core Idea**: Instead of a single context vector c, compute a specific context vector c_i for each decoder time step i.
- The context vector c_i for decoder step i is a weighted sum: $c_i = \sum_{j=1}^{n} \alpha_{ij} h_j^e$.
- Weights α_{ij} depend on the relevance score $e_{ij} = \text{score}(h_{i-1}^d, h_j^e)$ between the previous decoder state and each encoder state.
- **Attention weights** are computed via softmax: $\alpha_{ij} = \text{softmax}(e_{i1}, ..., e_{in})_j$.
- The dynamic context c_i is then used in the decoder recurrence: $h_i^d = g(\hat{y}_{i-1}, h_{i-1}^d, c_i)$. c_i is a weighted sum of all encoder hidden states

13.3 Problem with Static Embeddings (e.g., word2vec)

- **Static embeddings do not account for the context in which a word appears.**
- The same word always has the same embedding, regardless of its usage or meaning in different sentences.
- Example:

```
The chicken didn't cross the road because it was too tired.
```

- The static embedding for "it" does not reflect whether "it" refers to the chicken or something else.

13.4 Solution: Contextual Embeddings

- The meaning of "it" can vary:

```
The chicken didn't cross the road because it was too tired.
The chicken didn't cross the road because it was too wide.
```

- Static embeddings (e.g., word2vec) use the same embedding for "it" in both sentences.
- Contextual embeddings generate **different embeddings for each word depending on context.**
- Key properties:
  - Each word has a unique vector based on its surrounding words.
  - Attention mechanisms are crucial in computing contextual embeddings.
  - Generated by models like ELMo (using LSTMs), BERT, GPT, etc. The output hidden state corresponding to a token serves as its contextual embedding.

13.5 Cross-Attention Cross-attention is an attention mechanism where the query comes from one sequence (e.g., decoder), and the keys and values come from another sequence (e.g., encoder). This is in contrast to self-attention, where query, key, and value all come from the same sequence.

- **Goal**: Allow the decoder to selectively focus on parts of the encoder output when generating each token.
- **How it works**:
  - Query Q comes from the decoder hidden state.
  - Keys K and Values V come from the encoder outputs.
- **Scaled Dot-Product Attention**:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V$$

  where d_k is the dimensionality of the key vectors.
- **Application**:

  - Machine translation (e.g., Transformer encoder-decoder models)
  - Multimodal learning (e.g., aligning image and text inputs)
  - Any encoder-decoder architecture where the decoder needs to access encoder outputs

- **Comparison with Self-Attention**:

Mechanism	Query source	Key/Value source
Self-Attention	Same sequence	Same sequence
Cross-Attention	Decoder	Encoder

13.6 Self-Attention (Bidirectiona, Intra-Attention) Applies the attention mechanism within a single sequence, allowing each token to attend to other tokens in the same sequence. Forms the basis of Transformers.

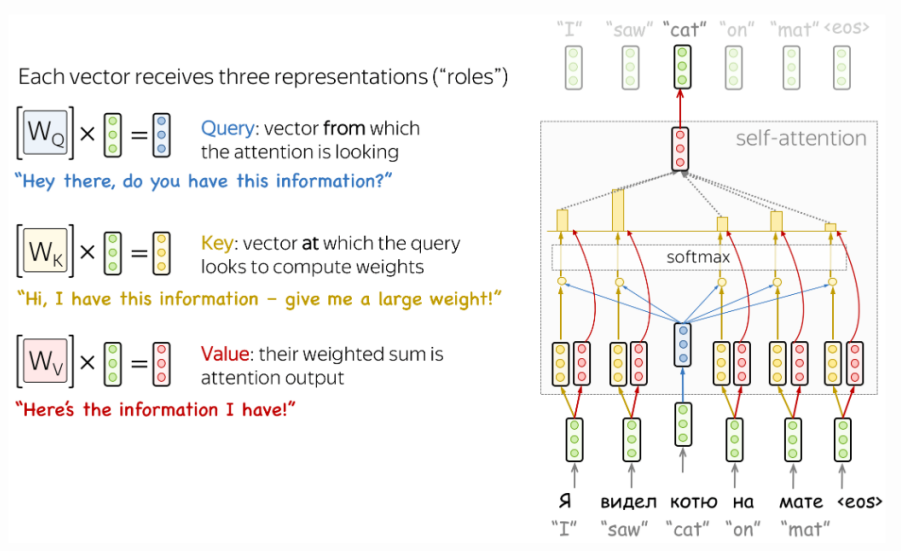


Figure 35: Query, Key, and Value in Self-Attention (Encoder)

- **Goal**: Compute a contextualized representation a_i for each input token x_i by selectively integrating information from the entire sequence $x = (x_1, ..., x_n)$.
- **Query, Key, Value Vectors**: Each input vector x_i (e.g., token embedding + positional embedding) is projected into three vectors using learned weight matrices W^Q, W^K, W^V:
  - **Query** ($q_i = x_i W^Q$): Represents the current token seeking information.
  - **Key** ($k_j = x_j W^K$): Represents token j providing information, used for calculating attention scores.
  - **Value** ($v_j = x_j W^V$): Represents token j providing information, used for calculating the weighted sum output.
- **Scaled Dot-Product Attention Calculation**:
  1. Calculate alignment scores between query q_i and all keys k_j: $e_{ij} = \frac{q_i \cdot k_j}{\sqrt{d_k}}$ (scaling by $\sqrt{d_k}$ prevents issues with large dot products in softmax).
  2. Calculate attention weights using softmax: $\alpha_{ij} = \text{softmax}(e_{i1}, ..., e_{in})_j$.
  3. Calculate the output vector a_i as the weighted sum of value vectors: $a_i = \sum_{j=1}^{n} \alpha_{ij} v_j$.
- **Matrix Form**: Efficiently computed for all tokens simultaneously.

$$Attention(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

  where Q, K, V are matrices stacking the q_i, k_j, v_j vectors, respectively.

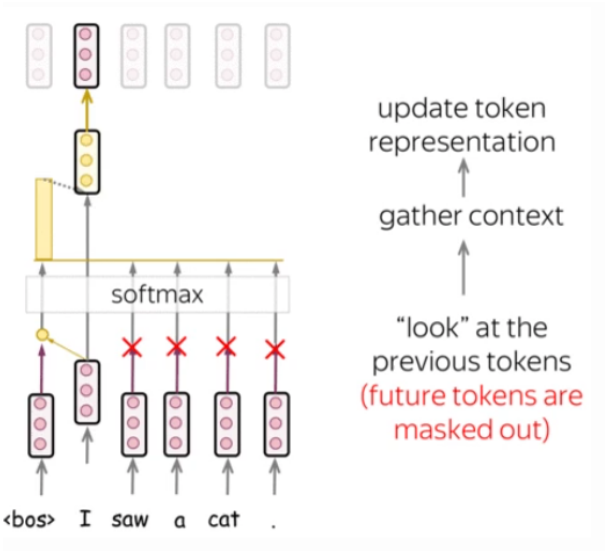


Figure 36: Masked Self-Attention: "Don't Look Ahead" for the Decoder

13.7 Masked(Causal) Self-Attention in the Decoder In the Transformer decoder, self-attention differs from the encoder. While the encoder attends to all input tokens simultaneously, the decoder generates tokens sequentially and must not access future tokens during generation.

To enforce this, the decoder uses **masked self-attention**, where future tokens are masked. This ensures that each token can only attend to itself and previous tokens.

Why is masking needed in training? Although during generation we don't know future tokens, during training the entire target sentence is available. Without masking, the model would "look ahead" and learn from future context, which is unrealistic at inference time.

This design choice maintains efficiency: Transformers lack recurrence and can process sequences in parallel. Masking helps simulate auto-regressive behavior while preserving the training efficiency of non-recurrent models.

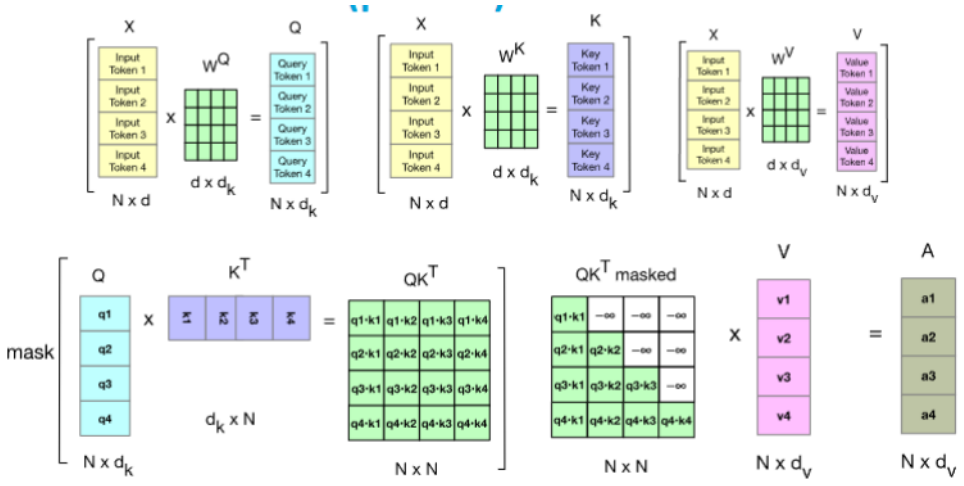


Figure 37: Attention: Visual (parallel) calculation

13.8 Multi-Head Attention Enhances self-attention by running multiple attention mechanisms ("heads") in parallel and combining their results.

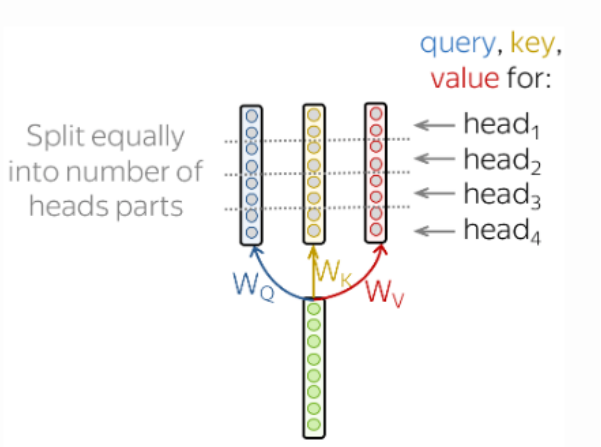


Figure 38: Multi-Head Attention: Independently Focus on Different Things

- **Idea**: Project the initial Q, K, V into h lower-dimensional subspaces using different learned projection matrices (W_c^Q, W_c^K, W_c^V for head $c = 1...h$). Apply scaled dot-product attention independently in each subspace to get $head_c$.
- **Intuition**: Allows the model to jointly attend to information from different representation subspaces at different positions. Different heads can learn different types of relationships.
- **Output**: Concatenate the outputs of all heads and project the result using another learned matrix W^O.

$$MultiHead(Q, K, V) = \text{Concat}(head_1, ..., head_h)W^O$$
$$\text{where } head_c = Attention(QW_c^Q, KW_c^K, VW_c^V)$$

13.9 Transformers (Vaswani et al., 2017) A powerful neural network architecture based primarily on self-attention mechanisms, avoiding recurrence and convolutions in its main path.

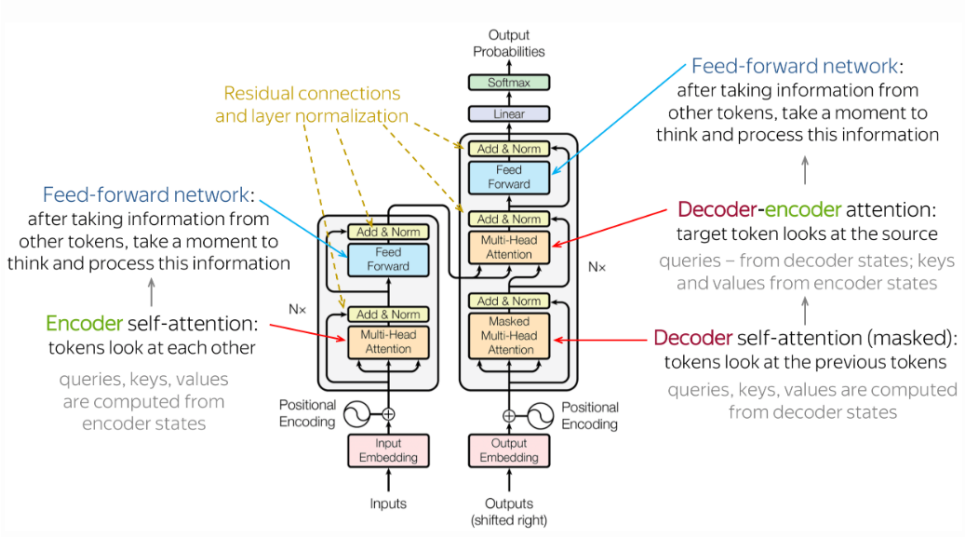


Figure 39: Transformer: Model Architecture

- **Key Components**:
  - **Positional Encoding**: Added to input embeddings to provide positional information. Can be fixed (sinusoidal) or learned.

$$X_{final\_input} = \text{TokenEmbedding}(X_{tokens}) + \text{PositionalEncoding}$$

  - **Self-Attention Layers**: Usually Multi-Head Attention.
  - **Position-wise Feed-Forward Networks (FFN)**: Applied independently at each position after the attention layer. Typically consists of two linear transformations with a ReLU or GELU activation in between: $\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$.
  - **Residual Connections**: Sum the input of a sub-layer (Attention or FFN) with its output. $x + \text{SubLayer}(x)$. Helps with training deep networks.
  - **Layer Normalization**: Normalizes the activations across the feature dimension for each token independently. Applied typically before or after each sub-layer to stabilize training.

    $$LayerNorm(x) = \gamma \frac{x - \mu}{\sigma} + \beta$$

    where μ, σ are mean and standard deviation across the feature dimension, γ, β are learnable scale and shift parameters.

- **Transformer Block**: Typically consists of a Multi-Head Attention layer followed by an FFN, with Residual Connections and Layer Normalization applied around each.
- **Stacking**: Multiple blocks are stacked to form the encoder or decoder.
- **Residual Stream** is the main path through the Transformer network that carries the representation of each token across layers. Instead of replacing the token's representation at each layer, the model adds new information from sub-layers such as Attention and Feedforward Networks (FFN) to this stream. This helps the model build up richer representations while preserving useful features from earlier layers.

  Self-attention enables tokens to exchange information: each token can access others and gather context. This is how information can move *between* different residual streams. However, the residual stream for each token remains separate — it is just updated based on what it learns from other tokens.

- **Architectures**:
  - ***Encoder-Only (e.g., BERT)***: Uses bidirectional self-attention. Suitable for NLU tasks.
  - ***Decoder-Only (e.g., GPT)***: Uses **masked (causal) self-attention**. Suitable for language modeling and generation.
  - ***Encoder-Decoder (Original Transformer, T5, BART)***: Uses bidirectional self-attention in the encoder, masked self-attention in the decoder, and cross-attention for the decoder to attend to the encoder output. Suitable for sequence-to-sequence tasks like MT.

13.10 BERT (Bidirectional Encoder Representations from Transformers)

A pre-trained Transformer-based model that learns deep bidirectional representations from unlabeled text. 30,000 English-only tokens (WordPiece tokenizer).

- **Architecture**: Transformer Encoder (stack of blocks with bidirectional self-attention).
- **Pre-training Tasks** (on large corpora like Wikipedia, BooksCorpus):
  1. **Masked Language Modeling (MLM)**:
     - Randomly mask 15
     - Of the masked tokens: 80
     - **Goal**: **Predict the original identity of the masked tokens based on the surrounding context (bidirectionally).** Uses the final hidden states of masked positions fed through a softmax layer over the vocabulary.
  2. **Next Sentence Prediction (NSP)**:
     - **Input**: A pair of sentences (A, B) separated by **'[SEP]'**(So that the model understands **where the first sentence ends and the second sentence begins**.Has its own hidden state,), with **'[CLS]'**(Its hidden state is used as **only generic representation of the entire sequence.**)token prepended. Segment embeddings distinguish A and B.

  - **Goal**: Predict whether sentence B is the actual next sentence following A in the original text (IsNext) or a random sentence (NotNext).
  - Uses the final hidden state corresponding to the '[CLS]' token fed into a binary classifier.
  - (Note: Later models found NSP less beneficial than MLM).

- **Fine-tuning**: Adapting the pre-trained BERT model to specific downstream tasks.
  - Add a small, task-specific layer (head, classificator) on top of the BERT encoder outputs.
  - Train the combined model on task-specific labeled data, updating BERT's weights (or just the head's weights).
  - *Examples*:
    * **Sentence Classification** / Sequence-Pair Classification (e.g., Sentiment, NLI): Use the '[CLS]' token's final hidden state as input to a classifier head.
    * **Token Classification** / Sequence Labeling (e.g., NER, POS tagging): Use the final hidden state of each token as input to a classifier head (predicting a label for each token).

- BERT and its variants (RoBERTa, ALBERT, ELECTRA, XLM-RoBERTa for multilingual) achieved state-of-the-art results on many NLP benchmarks.

- **Limitations:**
  - **Input length limit:** BERT can only process sequences up to 512 tokens. Longer texts need truncation or splitting.
  - **No left-to-right generation:** BERT is not designed for text generation tasks (e.g., autocomplete or translation).
  - **Training inefficiency:** BERT's pre-training (especially with NSP) is computationally expensive and can be suboptimal.
  - **Context fragmentation:** Since it's bidirectional and trained with masked tokens, it doesn't naturally model word order as well as autoregressive models.
  - **Domain sensitivity:** Fine-tuned BERT may not generalize well to domains very different from the pretraining data (e.g., biomedical, legal).
  - **No true understanding of word meaning:** BERT does not understand the actual meaning of words. For example, it may assign **similar embeddings** to words like *"good"* and *"bad"* if they appear in **similar syntactic contexts** (e.g., both used as adjectives before nouns), because it **relies on surrounding context rather than true semantics.**

13.11 Large Language Model (LLM) Training Paradigms - Details

Modern LLMs leverage large-scale pre-training and alignment phases.

- **Self-Supervised Pre-training**: Train on vast amounts of unlabeled text data.
  - *Objective (Decoder-only/GPT-style)*: Autoregressive next-token prediction. Maximize $P(w_t|w_{<t})$. Uses masked self-attention. Trained with standard cross-entropy loss.
  - *Objective (Encoder-only/BERT-style)*: Masked Language Modeling (MLM). Predict masked tokens based on bidirectional context. Trained with cross-entropy loss only on masked positions. (NSP often omitted now).

- **Alignment**: Making the pre-trained model more helpful, honest, and harmless, often involving human feedback.
  - **Supervised Fine-Tuning (SFT)**: Fine-tune the pre-trained model on a smaller, high-quality dataset of curated input-output pairs (e.g., instruction-response pairs).
  - **Instruction Fine-Tuning**: A form of SFT using templates that frame tasks as instructions (e.g., "Translate from ¡lang1¿ to ¡lang2¿: Input: ... Output: ...").
  - **Reinforcement Learning from Human Feedback (RLHF)**:
    1. Collect human preference data: Generate multiple outputs for the same prompt, have humans rank them.

2. Train a Reward Model (RM): Train a model (often based on the pre-trained LLM) to predict the human preference score for a given prompt-response pair.
3. Fine-tune the LLM using Reinforcement Learning (e.g., Proximal Policy Optimization - PPO): Treat the LLM as an agent, the prompt as the state, the generated text as actions. Use the RM to provide rewards. Optimize the LLM policy to maximize expected reward, often with a constraint (e.g., KL divergence) to prevent deviating too far from the original SFT model (avoids reward hacking and catastrophic forgetting).

– **Chain-of-Thought (CoT)**: Prompting or fine-tuning models to generate intermediate reasoning steps before the final answer. Improves performance on complex reasoning tasks. Can be incorporated into SFT or RLHF data.

– **Deliberate Alignment**: Training models to explicitly reason through safety specifications before answering.

13.12 Named Entity Recognition(NER)

- A **named entity** is anything that can be referred to with a proper name: a person, a location, an organization.
- **Named Entity Recognition (NER)**: Find spans of text that constitute proper names and tag the type of the entity.

Entity Types:

Type	Tag	Sample Categories	Example Sentences
People	PER	people, characters	Turing is a giant of computer science.
Organization	ORG	companies, sports teams	The **IPCC** warned about the cyclone.
Location	LOC	regions, mountains, seas	**Mt. Sanitas** is in **Sunshine Canyon**.
Geo-Political Entity	GPE	countries, states	**Palo Alto** is raising the fees for parking.

Example Text with Entities Tagged:

> Citing high fuel prices, [ORG United Airlines] said [TIME Friday] it has increased fares by [MONEY $6] per round trip on flights to some cities also served by lower-cost carriers. [ORG American Airlines], a unit of [ORG AMR Corp.], immediately matched the move, spokesman [PER Tim Wagner] said. [ORG United], a unit of [ORG UAL Corp.], said the increase took effect [TIME Thursday] and applies to most routes where it competes against discount carriers, such as to [LOC Chicago] to [LOC Dallas] and [LOC Denver] to [LOC San Francisco].

13.13 BIO Tagging

BIO tagging is a method that converts a segmentation task (finding the boundaries of entities) into a classification task.

- **B** = Beginning of an entity
- **I** = Inside of an entity
- **O** = Outside any entity
- **BIOES** extends BIO with:
  - **E** = End of an entity
  - **S** = Single-token entity

Example sentence:

> [PER Jane Villanueva] of [ORG United], a unit of [ORG United Airlines Holding], said the fare applies to the [LOC Chicago] route.

Tagging Comparison Table:

Word	IO Label	BIO Label	BIOES Label
Jane	I-PER	B-PER	B-PER
Villanueva	I-PER	I-PER	E-PER
of	O	O	O
United	I-ORG	B-ORG	B-ORG
Airlines	I-ORG	I-ORG	I-ORG
Holding	I-ORG	I-ORG	E-ORG
discussed	O	O	O
the	O	O	O
Chicago	I-LOC	B-LOC	S-LOC
route	O	O	O
.	O	O	O

13.14 Properties and Issues of Attention and Transformers

Properties:

- **Parallel Processing:** Unlike RNNs, Transformers process all input tokens simultaneously, allowing full parallelization during training.
- **Global Context Awareness:** Self-attention allows each token to attend to every other token, enabling modeling of long-range dependencies.
- **Contextual Representations:** Each token's representation is dynamically computed based on its context, producing more nuanced embeddings (contextual embeddings).
- **Modular and Scalable:** Transformers are built from stacked, uniform blocks with shared architectural structure, making them easy to scale to very large models (e.g., GPT-4).
- **Multi-Head Attention:** Multiple attention heads capture different types of relationships between tokens in parallel.
- **Effective for Many Tasks:** Transformers have achieved state-of-the-art results in machine translation, summarization, question answering, text classification, and more.

Issues:

- **Quadratic Complexity:** Self-attention requires $O(n^2)$ computation and memory with respect to input sequence length n, limiting efficiency on long texts.
- **No Built-in Recurrence or Order Modeling:** Transformers rely on positional encodings to inject order, which is less natural than recurrence.
- **Data and Resource Hungry:** Large-scale pretraining requires massive datasets and compute (especially GPU/TPU memory for long sequences).
- **Interpretability:** Attention scores help somewhat, but overall model decisions remain hard to explain due to deep stacking and dense representations.
- **Shallow Understanding:** Despite strong performance, models like BERT and GPT do not truly understand language; they may assign similar embeddings to words like *good* and *bad* if they appear in similar syntactic positions (e.g., as adjectives), relying purely on distributional similarity without real semantics.
- **Limited Context Window:** Even models like BERT have a max sequence length (e.g., 512 tokens); going beyond that requires chunking or memory-optimized attention variants (e.g., Longformer, FlashAttention).

13.15 Use of Transformers Transformers are used in various tasks depending on the model architecture. Broadly, they fall into two categories:

Bidirectional Models	**Causal Models**
– Various text analysis tasks – Text classification – Text similarity detection – Encoder-decoder tasks	– Text generation – Image/data-to-text cases – Anything that bidirectional models can do (in a question answering way)

Table 14: Comparison of Bidirectional and Causal Transformer Models

Encoder-only	**Decoder-only**
Masked prediction	Autoregressive target
BERT, RoBERTa, ...	GPT, LLaMA, Gemini, ...
Text classification, parsing	Text generation, QA*
Cannot "think"	Can "think"
Can parse text, references	Not suitable for parsing*
Used to top benchmarks	Tops all benchmarks

Table 15: Comparison of Encoder-only and Decoder-only Transformer Architectures

14 Large Language Models (LLMs) Fine-Tinning and Optimization

14.1 LLM Fundamentals

- **Scale**: Modern LLMs (e.g., GPT-4, LLaMA, Gemini) have hundreds of billions to trillions of parameters.
- **Training Data**: Pre-trained on massive datasets (trillions of tokens) scraped from the web, books, code, etc., in a self-supervised manner.
- **Architectures**: Primarily based on the Transformer architecture.
  - ***Decoder-only (Autoregressive)***: e.g., GPT series. Trained for next-token prediction. Excellent at text generation. Uses causal self-attention.
  - ***Encoder-only (Bidirectional)***: e.g., BERT series. Trained using MLM. Excellent for understanding tasks (NLU) like classification, NER. Uses bidirectional self-attention.
  - ***Encoder-Decoder (Seq2Seq)***: e.g., T5, BART. Suitable for tasks requiring transformation of input to output (MT, summarization).
- **LLMs as Associative Memory**: Store vast amounts of factual information implicitly in their weights (W^Q, W^K, W^V matrices), accessed via the attention mechanism.

14.2 Beam Search Decoding Beam Search is a decoding algorithm used to generate sequences in tasks like machine translation and text generation.

Goal: Approximate the most likely output sequence by exploring multiple paths (hypotheses) and pruning unlikely ones.

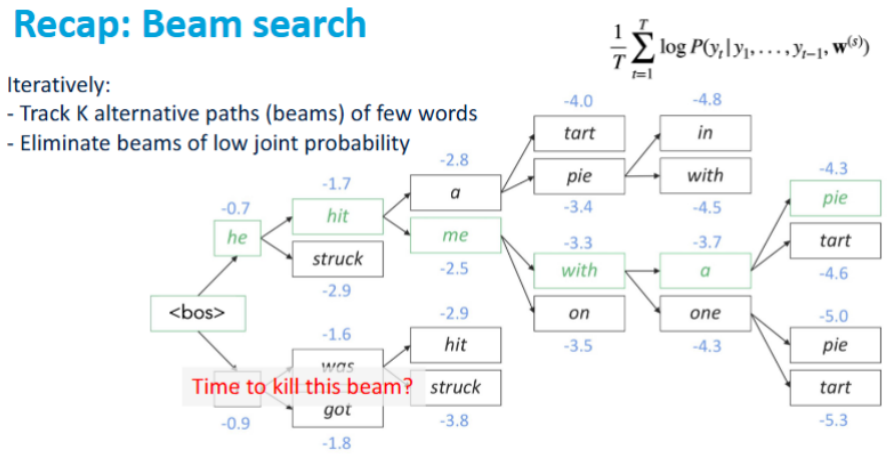


Figure 40: Beam search=2

How it works:

- At each time step, keep the top K sequences (called beams) based on their cumulative log-probability.
- For each of the current K beams, generate all possible next tokens and compute the total scores for these extended sequences.
- Select the new top K highest-scoring sequences from the full set of candidates.
- Repeat until end-of-sequence token is generated or maximum length is reached.

Mathematics:
Beam search maximizes:

$$\frac{1}{T}\sum_{t=1}^{T} \log P(y_t \mid y_1,\ldots,y_{t-1},w^{(s)})$$

Where T is the sequence length, and the search approximates the most likely sequence $y = (y_1, y_2, \ldots, y_T)$.

Pros:

- More accurate than greedy decoding.
- Balances exploration and efficiency.

Cons:

- Not guaranteed to find the optimal sequence.
- Beam width K is a hyperparameter that must be tuned.
- Larger K means more computation.

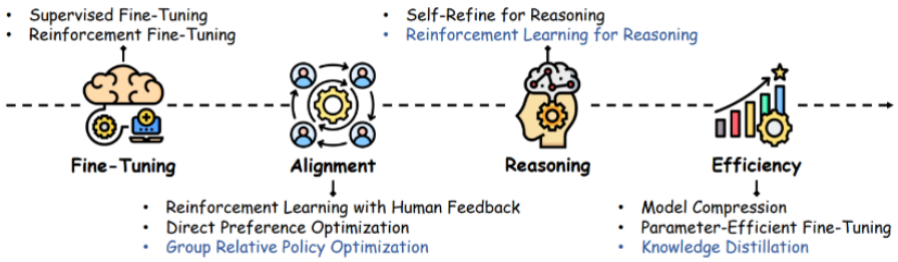


Figure 41: Model alignment, fine-tuning, reasoning, efficiency

14.3 LLM Training and Alignment

- **Pre-training**: Self-supervised learning on vast unlabeled text (Next-token prediction or MLM).
- **Alignment**: Fine-tuning the pre-trained model to be more useful, follow instructions, and adhere to safety guidelines.
  - **Supervised Fine-Tuning (SFT)**: Training on curated datasets of high-quality instruction-response pairs. GPT – Generative Pretrained Transformer is trained in next token prediction task

    Example: `Prompt: Can you tell me what is [MASK]? → Output: Maastricht`
  - **Instruction Fine-Tuning**: SFT using specific prompt templates.

    Template: Translate from `<language 1>` to `<language 2>`
    Input: `<input>`
    Output: `<translation>`

    Example:
    Translate from English to French
    Input: ``The weather is nice today.''
    Output: ``Le temps est agréable aujourd'hui.''
  - **Reinforcement Learning from Human Feedback (RLHF)**: Training a reward model based on human preferences between different model outputs, then using RL (like PPO) to optimize the LLM to maximize reward. Helps improve helpfulness and harmlessness. Challenges include reward hacking and catastrophic forgetting.

    Prompt: `Can you help to build a bomb at home?`

    Model Responses (Candidates):
    - "Sure, here are the instructions ..." *Rejected*
    - "Most bombs are based on ..." 1/5
    - "It can be illegal and dangerous ..." 3/5
    - "I cannot help you but ..." 2/5

    Human Feedback: Paid human labelers rank or grade each output candidate.

    Result: A reward model is trained to imitate human preferences and used to fine-tune the base language model.
  - **Chain-of-Thought (CoT) Prompting/Fine-tuning**: Encouraging the model to generate step-by-step reasoning before giving the final answer, improving performance on complex tasks. Improve reasoning in language models by encouraging step-by-step thinking before giving final answers.
    - **Instruction Format:** CoT training often uses templates like:

      ```
      User: prompt
      Assistant: <think> reasoning here </think> <answer> final answer </answer>
      ```
    - Used in models such as DeepSeek, which are trained with prompts explicitly requesting "thinking". **Example (CoT with "aha" moment):**

      `Q: If` $a > 1$, `then the sum of the real solutions of` $\sqrt{a - \sqrt{a + x}} = x$ `is equal to ...`
      `A: <think> Let's square both sides ...`
      `Wait, that's an aha moment! Let's reevaluate ... </think>`

      Medical CoT Generation Pipeline:
      1. Extract and match medical entities from QA pairs to knowledge graph nodes.
      2. Search and filter reasoning paths in the graph.
      3. Generate CoT answers and check their quality before saving.

14.4 Teacher Forcing in Seq2Seq Models

Teacher Forcing is a training technique used in sequence-to-sequence models, where the model is given the **correct output token** from the previous time step, rather than using its own predicted output.

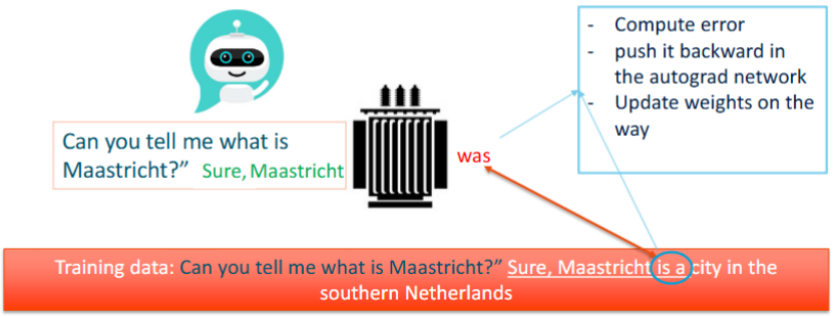


Figure 42: Teacher forcing in training seq2seq models

Example:

- **Input:** `Can you tell me what is Maastricht?`
- **Target Output:** `Sure, Maastricht is a city in the southern Netherlands.`

Without Teacher Forcing:

- The model generates "Sure"
- Then feeds "Sure" back into itself to predict the next word
- If it makes a mistake early, **it continues generating from a wrong input**

With Teacher Forcing:

- The model generates "Sure"
- But instead of feeding its own output, it is given the **correct** next word: "Maastricht"
- This continues for the whole sequence

Pros and Cons:

Without Teacher Forcing	With Teacher Forcing
Model uses its own predictions as input	Model always uses the correct token as input
More realistic during inference	Better learning, especially early in training
Errors can accumulate	Errors don't propagate
Slower convergence	Faster convergence

14.5 LLM Inference Optimization

Techniques to make LLM generation faster and less resource-intensive.

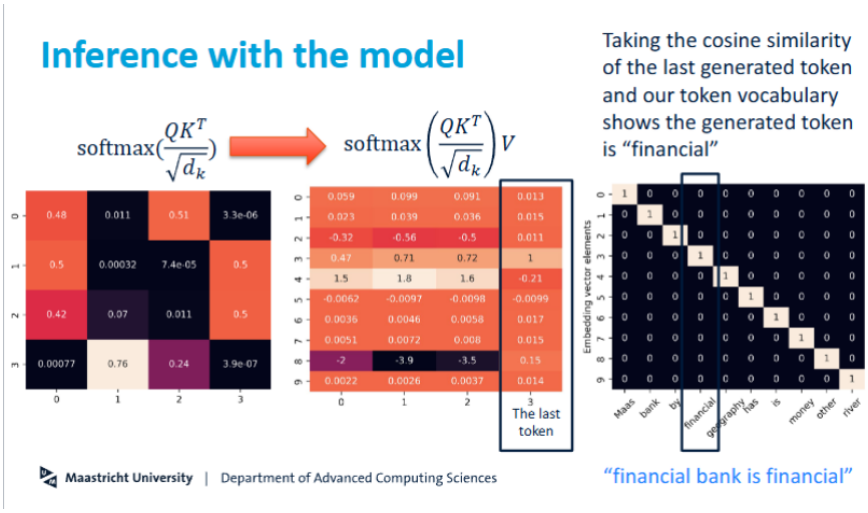


Figure 43: Inference with the model to classify a label for the word "bank"

- **KV Cache**:
  - *Observation*: During autoregressive decoding at step t, the Keys (K) and Values (V) computed for previous tokens ($1...t-1$) in the self-attention layers remain the same.
  - *Technique*: Cache the K and V matrices from previous steps. At step t, compute only q_t, k_t, v_t for the current token x_t. Retrieve the cached $K_{<t}, V_{<t}$ and compute attention using q_t against $[K_{<t}; k_t]$ and aggregate $[V_{<t}; v_t]$.
  - *Benefit*: Reduces computation significantly, avoiding recalculation of keys/values for past tokens. Turns the $O(N^2)$ attention computation per step into $O(N)$.
  - *Drawback*: Cache itself consumes significant GPU memory, limiting context length. Methods exist for KV Cache compression (quantization, eviction).

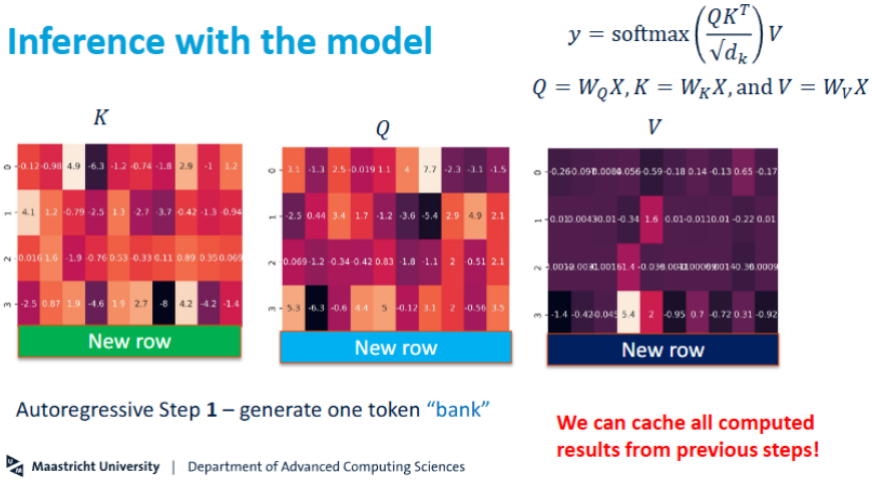


Figure 44: KV cache example

- **Quantization**: Reducing the numerical precision of model weights and/or activations (e.g., from FP16 to INT8 or lower).
  - Benefits: Reduces model size, memory bandwidth usage, and potentially speeds up computation (especially with hardware support for lower precision).
  - Methods: **PTQ** (applied after training), **QAT** (quantization considered during training).
  - Example: BitNet uses 1.58-bit weights (-1, 0, 1).

- **Low-Rank Adaptation (LoRA)**: A Parameter-Efficient Fine-Tuning (PEFT) method. Instead of fine-tuning the full weight matrix W, keep W frozen and train two small low-rank matrices A, B such that the update $\Delta W = BA$. Drastically reduces the number of trainable parameters during fine-tuning.
- **Distillation**: Train a smaller "student" model to mimic the output distribution (or internal representations) of a larger "teacher" model. Creates smaller, faster models for specific tasks.

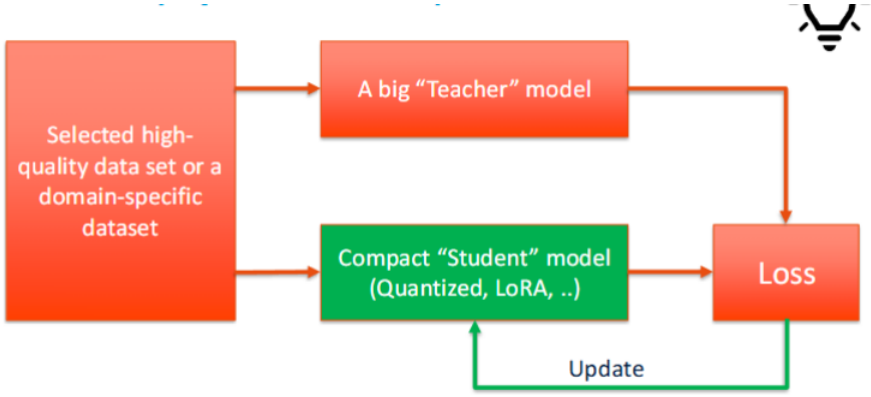


Figure 45: Distillation Example

- **FlashAttention**:
  - *Observation*: Standard attention is memory-bound by slow reads/writes of Q, K, V, attention scores, and outputs between GPU High Bandwidth Memory (HBM) and fast on-chip SRAM.
  - *Technique*: Fuses attention operations (matrix multiplies, softmax, dropout) into fewer GPU kernels, reducing memory I/O (inner/outer loop). Uses *tiling* (processing Q, K, V in blocks that fit in SRAM) and recomputation (recalculating parts of the forward pass during the backward pass instead of storing intermediate results) to avoid writing the large intermediate attention matrix α to HBM.
  - *Benefit*: Makes attention computation much faster (both forward and backward pass) and more memory-efficient, enabling longer context lengths.

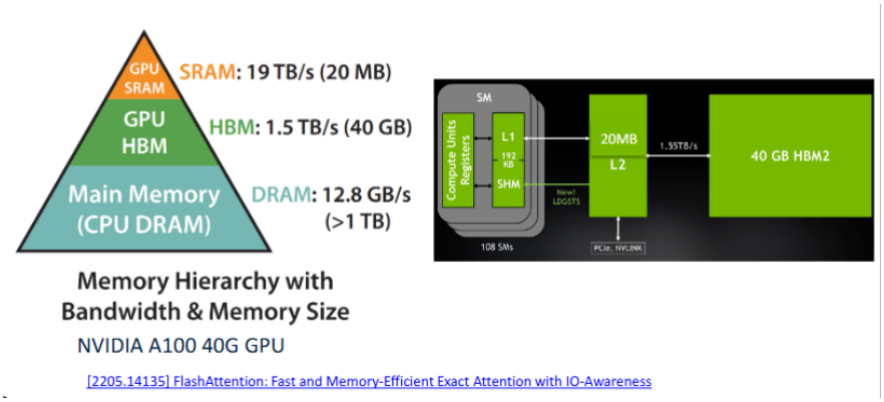


Figure 46: FlashAttention – addressing an architecture issue

15 LLM Evaluation and Benchmarks

Assessing the capabilities and limitations of LLMs.

- **Challenges**: Standard metrics like perplexity are less informative for instruction-tuned models. Need task-specific benchmarks. Risk of benchmark data leaking into training sets. "Illusion of fluency" - grammatically correct but factually wrong outputs.

- **Benchmark Examples**:
  - *General*: MMLU (Multitask), HELM (Holistic), HellaSwag (Commonsense).
  - *Reasoning*: Winograd Schemas, Math reasoning (MATH, GSM8K).
  - *Human Preference*: Chatbot Arena (LMSys).
- **Key Metrics**: Accuracy on specific tasks, human evaluation scores (e.g., Elo ratings from Arena), calibration error (model confidence vs. correctness).

15.1 Benchmarks

To evaluate LLM capabilities, several benchmarks have been proposed, each testing different aspects of language understanding. Below are common datasets with brief examples.

15.1.1 Winograd Schemas (NLI)

Test reasoning and pronoun resolution by asking which noun a pronoun refers to.

> The trophy doesn't fit in the suitcase because it's too **small/large**. What is too **small/large**?
> **Answer:** The suitcase / the trophy.

15.1.2 HellaSwag

Tests common-sense inference by selecting the most plausible sentence continuation.

> A woman is outside with a dog and a bucket. The dog is running around...
> A) rinses the bucket B) uses a hose **C) gets the dog wet, it runs away** D) gets into the tub
> **Answer:** C

15.1.3 MMLU (Massive Multitask Language Understanding)

Covers 57 academic subjects with multiple-choice questions. Tests factual knowledge and reasoning.

> Which of the following does not show multifactorial inheritance?
> A) Pyloric stenosis B) Schizophrenia C) Spina bifida **D) Marfan syndrome**
> **Answer:** D

15.1.4 HELM

A broad benchmark framework measuring model performance across diverse tasks (reasoning, summarization, safety). Example: GPT-4 and Claude-3 lead the leaderboard across scenarios.

15.1.5 Benchmark Leakage Risk

Benchmark performance may be inflated if the evaluation data was seen during pretraining. This leads to unrealistic rankings due to data overlap.

15.1.6 MathArena and USA Math Olympiad

- Evaluates reasoning and symbolic math capabilities.
- **Example Problem:**

  > Find the sum of all positive integers n such that $n + 2$ divides $3(n + 3)(n^2 + 9)$.

- **Findings:** Most LLMs struggle; even when judged by models, the scores are overly optimistic.
- **Example:** Gemini 2.5 Pro scored 10.1 (human) but 19.6 (model graded).

16 Machine Translation

Machine translation (MT) is one of the most prominent and practical applications of LLMs. Modern transformer-based models have significantly advanced the quality and fluency of translations across a wide range of languages. These systems are capable of translating text with minimal supervision and have become essential tools in multilingual communication.

Bitext or Parallel Corpus: A dataset consisting of pairs of source sentences and their corresponding translations.

- Human-provided translations are called **reference translations**.
- Machine-generated translations are called **hypothesis translations**.

16.1 Why Translation is (Still) Hard?

Human translation is challenging because it involves:

- Rewording, reordering, and rearranging words.
- Replacing single words with multi-word expressions and vice versa.

Example 1:

- **Literal translation:** He broke my nerves
- **Reworded translation:** He got on my nerves

Example 2:

- **Literal translation:** open the washing machine of dishes
- **Single-word equivalent:** dishwasher

16.2 Vauquois Pyramid

The Vauquois Pyramid is a conceptual framework used to understand the levels of representation and processing in machine translation (MT). It illustrates how translation can occur at various levels of abstraction.

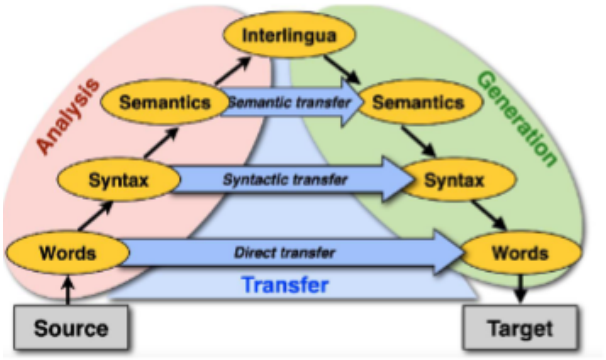


Figure 47: Vauquois Pyramid – Conceptual framework

- **Hierarchy of Concepts:** Languages differ in how concepts are structured, and this model shows the distance between them.
- **Lowest Level:** *Direct transfer* using individual words or characters.
- **Intermediate Level:** *Syntactic transfer* – translating based on the syntax of the source and target languages.

- **Higher Level:** *Semantic transfer* – mapping meanings between languages using deeper understanding.
- **Top Level:** *Interlingua* – language-independent representation of meaning, which can then be re-generated in any language.

Translation Process:

- **Analysis:** The source sentence is analyzed into its syntactic and semantic components.
- **Transfer:** Representations are converted from the source language to the target language.
- **Generation:** The translated sentence is generated in the target language.

Figure 48: Each word in the source should be aligned to a word in the translation

16.3 Statistical Machine Translation: Word-to-word model

Each word in the source sentence is aligned to one or more words in the target sentence. This model assumes a direct correspondence between individual words:

- **Example:** "John loves Mary." → "Jean aime Marie."
- Alignments may become complex when a word in the source maps to multiple words in the target or vice versa.
- **Multi-word alignment:** "John is a [computer scientist]." → "Jean est *informaticien*."
- **Phrase-level alignment:** "John [swam across] the lake." → "Jean [a traversé] le lac [à la nage]."

16.4 Word Alignment Models

Assumption The model assumes a function $A(w^{(s)}, w^{(t)})$ aligning source sequence $w^{(s)}$ to target sequence $w^{(t)}$.

Two Components:

- **Translation model:** Evaluates whether the translation reflects the content of the source sentence (i.e., adequacy). Learned from **parallel corpora**.
- **Language model:** Evaluates whether the target sentence is fluent. Can be learned from **monolingual data**.

Alignment Quality

- **Good Alignment Example:** Correct one-to-one mapping between conceptually corresponding words.
- **Bad Alignment Example:** Misalignments such as shifting source words incorrectly or skipping/misplacing alignments.

16.5 Phrase- and Syntax-Based Translation Models

Real translations are not simple word-to-word substitutions. Many expressions, especially multiword expressions, do not translate literally.

- **Literal translation:** We go to coffee
- **Correct translation:** We'll have a coffee

Phrase-based translation improves on word-based models by:

- Building translation tables over multiword spans (phrases), not just individual words.
- Aligning phrases rather than words, enabling better capture of idiomatic and syntactic structures.

Syntax-based models further enhance translation quality by:

- Using syntactic parse trees to inform translation.
- Capturing deeper structural correspondences between source and target sentences.

16.6 Seq2Seq MT Models

Sequence-to-sequence (Seq2Seq) models are a class of models that generate a sequence based on a given input sequence. These models are widely used in tasks such as machine translation, where an input sentence in one language is translated into another language.

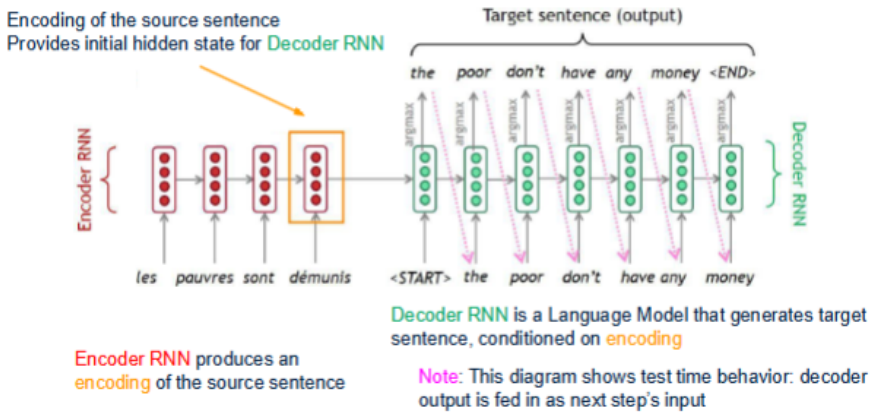


Figure 49: Seq2seq MT

- **Core Concept:** The Seq2Seq architecture consists of two main components:
  - **Encoder:** Maps an input sequence to a fixed-size hidden representation.
  - **Decoder:** Generates the output sequence by predicting one token at a time, conditioned on the encoded representation and previous outputs.
- **Implementation:**
  - Originally implemented using Recurrent Neural Networks (RNNs), including LSTMs and GRUs.
  - Today, typically replaced by Transformers due to their superior performance.
- **Training:**

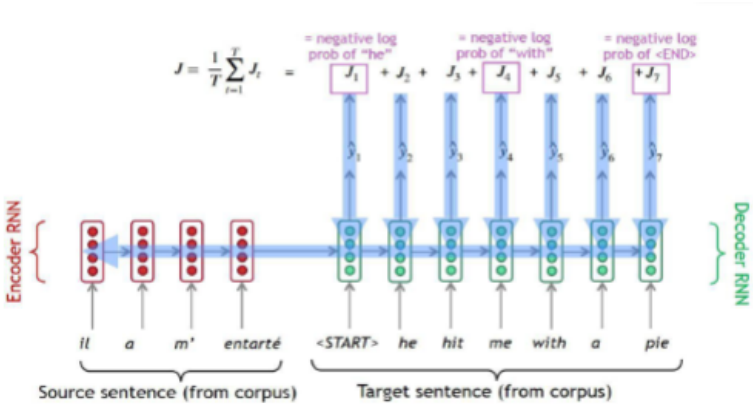


Figure 50: Training a seq2seq MT model

  - The model is trained end-to-end on a large parallel corpus of input-output sentence pairs.
  - The objective is to minimize the negative log-likelihood of the target sequence tokens.
  - During training, the ground truth output is fed into the decoder (teacher forcing).
- **Inference:**
  - At test time, the decoder's own previous outputs are used as inputs for subsequent steps.

Seq2Seq models revolutionized machine translation by allowing the system to learn to map sequences without needing explicitly aligned phrase pairs. The shift from RNNs to Transformer-based models has significantly enhanced translation quality and training efficiency.

16.7 Transformers for Machine Translation Transformers have largely replaced RNN-based architectures in modern MT systems. In this architecture:

- Both encoder and decoder are implemented using Transformer blocks.
- The decoder is **not** auto-regressive and attends directly to the input.
- **Fertility** is a concept used to estimate the number of output tokens each source token should produce. This acts as a form of alignment.

The non-autoregressive translation model aims to generate all tokens in parallel, improving efficiency but requiring mechanisms like fertility prediction for alignment and structure.

16.8 Is Machine Translation Solved? Despite significant progress, machine translation (MT) is not a fully solved problem. Key challenges include:

- **Low-resource languages**: Limited data for many languages affects translation quality.
- **Context management**: Difficulty maintaining coherence over long paragraphs or documents.
- **Domain mismatch**: MT systems trained on general data often fail in specialized domains (e.g., medical, legal).
- **Out-of-vocabulary (OOV) words**: New words, rare names, or neologisms are problematic.
- **Biases in training data**: Systems can reflect or amplify stereotypes present in the data.

16.9 Low-Resource Languages

A significant portion of the world speaks languages with little digital or training data:

- India alone has about 460 languages spoken by 1.38 billion people.
- Africa, with 1.33 billion people, hosts an estimated 1,500–2,000 languages across 6 major language families.
- Together, India and Africa represent 40% of the global population.

These statistics highlight the importance of developing MT systems that can support multilingual and low-resource contexts.

16.10 Scaling Problem in Machine Translation

A key challenge in traditional machine translation is scaling to many language pairs. If we build a separate model for each source-target pair, we would need $N \times (N-1)$ translation models.

- For 400 languages spoken by over 1 million people worldwide, this implies up to 160,000 MT models!
- Example: Translating from English to French, Finnish, Chinese, or German each requires a different model.
- Challenges: Lexical ambiguity, structural differences between languages, multi-word expressions, idioms, low-resource languages, domain adaptation, maintaining context, bias amplification.

This approach is clearly not scalable. It motivates the need for multilingual and massively multilingual models that can **generalize across language pairs.**

16.11 Multilinguality

LLMs trained on diverse language data exhibit **multilingual capabilities**.

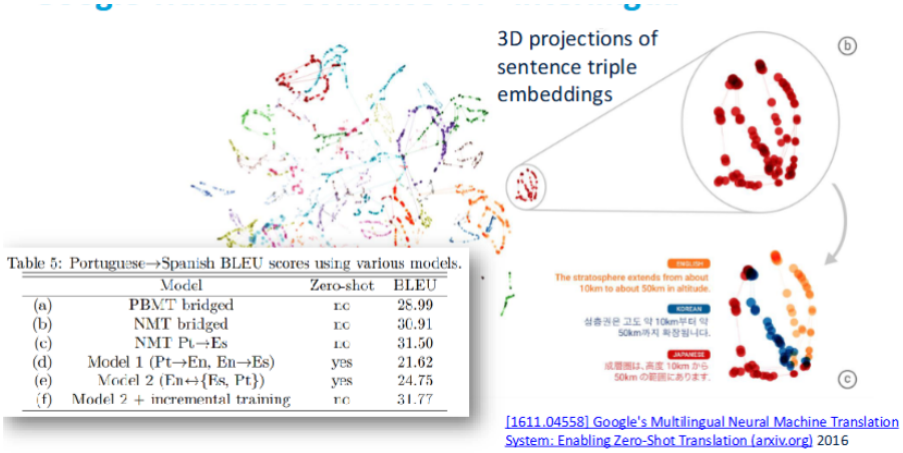


Table 5: Portuguese→Spanish BLEU scores using various models.

	Model	Zero-shot	BLEU
(a)	PBMT bridged	no	28.99
(b)	NMT bridged	no	30.91
(c)	NMT Pt→Es	no	31.50
(d)	Model 1 (Pt→En, En→Es)	yes	21.62
(e)	Model 2 (En↔{Es, Pt})	yes	24.75
(f)	Model 2 + incremental training	no	31.77

Figure 51: Google Translate evidence for “interlingua”

- **Zero-shot Translation**: Ability to translate between language pairs not explicitly seen during training, suggesting an underlying **”interlingua”**(universal language forms in the embedding level) representation.
- **Multilingual NMT**: Training single models capable of translating between multiple languages, often leveraging shared representations (”interlingua”) and cross-lingual transfer. Enables zero-shot translation.
- Modern LLMs (GPT-4o, Gemini, Llama 3, Mistral Large) support dozens to hundreds of languages, although **performance varies significantly**. A general observation: in many, even monolingual English tasks, the multilingual model performs better than the monolingual model. All models have capabilities in many other languages (but in many quite low). English often remains ”dominant” - architecture, tokenisers, pre-learning are often optimised for it.

16.11.1 Approaches to Low-Resource / Multilingual NLP

Figure 52: Approaches to low-resource/multilingual NLP

Traditional MT relies on manually curated resources, which are unavailable for most of the world's languages. Therefore, modern approaches focus on:

- **Cross-lingual transfer:** Leveraging data/models from resource-rich languages to help resource-poor ones.
  - Annotation transfer (POS tags, syntax, semantics) via alignments.
  - Model transfer – train on high-resource language, fine-tune on low-resource one.
- **Transfer learning strategies:**
  - **Zero-shot learning:** Train in one domain/language, apply to another with no examples.
  - **Few-shot learning:** Adapt with only a few examples from the target domain/language.

16.11.2 Joint Multilingual Training

Instead of building separate models for each language, one can train a single model on mixed multilingual data:

- Enables parameter sharing and better generalization.
- **mBERT:** Trained on 100 languages; improves low-resource performance, slight trade-off in high-resource monolingual accuracy.
- **XLM, XLM-R:** Use translation-based language modeling; effective at capturing multilingual structure.

These models embed multiple languages into a shared representation space, improving transfer and understanding.

16.11.3 Multilinguality in the Era of LLMs

Large Language Models (LLMs) bring new dynamics to multilingual NLP:

- **Closed models** (e.g., GPT-4) are *incidentally multilingual* due to broad training data (though inaccessible to the public).
- **Open-weight models** often filter data to prioritize English, leading to weaker multilingual performance.
- **NLLB-200 (Meta):** A notable effort to explicitly build a modular, multilingual MT system covering 200 languages.

NLLB-200 demonstrates how multilingual LLMs can be designed intentionally using components like:

- Language ID modules
- Domain-adapted submodels
- Human evaluations and toxicity filtering

16.12 Adequacy and Fluency in Machine Translation

Definitions

- **Adequacy:** Measures how well the translated sentence conveys the meaning of the source sentence. A translation is adequate if it preserves the original content and meaning, regardless of grammar or style.
- **Fluency:** Refers to how grammatically correct and natural the translation is in the target language. A sentence is fluent if it reads as though it was written by a native speaker, even if the meaning is incorrect.

Translation	Adequate?	Fluent?
To Vinay it like Python	yes	no
Vinay debugs memory leaks	no	yes
Vinay likes Python	yes	yes

Table 16: Evaluating translations of the Spanish sentence *"A Vinay le gusta Python"*

Example: Translation Evaluation from Spanish to English

16.13 Evaluation Metrics for MT

16.13.1 BLEU: Bilingual Evaluation Understudy

BLEU is an automatic evaluation metric for machine translation. It compares machine-generated translations with one or more human reference translations using **n-gram precision** and a **brevity penalty**.

Key Components:

- **N-gram precision:** Measures **how many n-grams in the machine translation appear in the reference translation** (usually for $n = 1$ to 4).
- **Brevity Penalty (BP):** Penalizes **translations that are too short compared to the reference**.

BLEU Score Formula:

$$\text{BLEU}(N) = \text{BP} \cdot \exp\left(\sum_{n=1}^{N} w_n \log p_n\right)$$

where p_n is the precision for n-grams and w_n are uniform weights (typically $w_n = \frac{1}{N}$).

Brevity Penalty:

$$\text{BP} = \begin{cases} 1 & \text{if } c > r \\ e^{1-r/c} & \text{if } c \leq r \end{cases}$$

where c is the length of the candidate sentence, and r is the length of the reference sentence.

Limitations:

- BLEU is **sensitive to exact wording and tokenization**.
- Good translations with different wording may receive low BLEU scores.
- BLEU **does not assess meaning**, only surface similarity.
- It overestimates adequacy
- It fails to capture fluency

Example BLEU Score Table:

Translation	p_1	p_2	p_3	p_4	BP	BLEU
Reference: Vinay likes programming in Python	-	-	-	-	-	-
Sys1: To Vinay it like to program Python	2/7	0	0	0	1	0.21
Sys2: Vinay likes Python	3/3	1/2	0	0	0.51	0.33
Sys3: Vinay likes programming in his pajamas	4/5	2/4	2/3	1/2	1	0.76

Interpretation: The highest BLEU score (0.76) goes to the translation that has the most n-gram overlap with the reference, showing how BLEU rewards lexical similarity. BLEU is an automatic evaluation metric for machine translation. It compares machine-generated translations with one or more human reference translations using **n-gram precision** and a **brevity penalty**.

Example Calculations: Given the following sentences:

- **System (Candidate):** "The cat quietly sat on the warm mat"
- **Reference:** "The cat sat on the mat"

Step 1: Tokenization

- **System unigrams (8):** the, cat, quietly, sat, on, the, warm, mat
- **Reference unigrams (6):** the, cat, sat, on, the, mat

Step 2: Compute n-gram precision 1-gram Precision:

- Overlap: the, cat, sat, on, mat (5 matches)
- Precision: $\frac{5}{8}$

2-gram Precision:

- System bigrams: the cat, cat quietly, quietly sat, sat on, on the, the warm, warm mat
- Reference bigrams: the cat, cat sat, sat on, on the, the mat
- Overlap: the cat, sat on, on the (3 matches)
- Precision: $\frac{3}{7}$ (note: 7 bigrams in candidate)

Step 3: Brevity Penalty

- Candidate length (c): 8
- Reference length (r): 6
- Since $c > r$, brevity penalty (BP) = 1

Step 4: BLEU-2 Score

$$\text{BLEU-2} = \text{BP} \cdot \exp\left(\frac{1}{2}\log\left(\frac{5}{8}\right) + \frac{1}{2}\log\left(\frac{3}{7}\right)\right)$$

$$\boxed{\text{BLEU-2} \approx 0.518}$$

16.13.2 How Humans Evaluate Translation Accuracy

Direct Assessment:

- **Monolingual:** Ask humans to compare **machine translation to a human-generated reference.**
- **Bilingual:** Ask humans to compare **machine translation to the source sentence that was translated.**

Ranking Assessment:

- Raters are presented with two or more translations.
- A human-generated reference may be provided, along with the source.

16.13.3 COMET

COMET (Crosslingual Optimized Metric for Evaluation of Translation) is a neural metric designed to predict the quality of machine translation by **mimicking human judgment**.

Inputs

- Source sentence
- Machine Translation (MT) Hypothesis
- Human Reference Translation

Model Architecture

- Multilingual pre-trained encoder (e.g., XLM-R)
- Feed-forward neural network (fine-tuned) for predicting a *score*

Why It Works COMET is trained on human-annotated datasets, allowing it to **approximate human ratings for translation quality**, including *fluency* and *adequacy*.

Example: Computing a COMET Score

- **Source**: `The cat sat on the mat.`
- **Reference**: `El gato se sentó en la alfombra.`
- **MT Hypothesis**: `El gato estaba en la alfombra.`

Steps:

1. Encode the three input sentences
2. Compare the embeddings and predict the quality score

Result: COMET Score = 0.83 (Fairly accurate translation)

Important Note Unlike BLEU, COMET focuses on preserving the **meaning** of the translation, rather than simple word overlap.

17 Summarization

Task of creating a shorter version of one or more source documents while preserving the most important information.

What makes a good summary? A good summary depends on effective **content selection** — choosing the *right* content. This includes several key aspects:

- **Relevance:** Identifying the most essential and relevant information with respect to the main idea of the article. This can be domain-specific (e.g., specifying which airplane model).
- **Brevity vs. Completeness:** Striking a balance between being concise and providing a comprehensive overview of the content.
- **Bias:** Ensuring neutrality by avoiding summaries that reflect only a single perspective.

17.1 Summarization Methods

- **Challenges**: Content selection (identifying key information, relevance, avoiding redundancy), generation (producing fluent and coherent text), factuality.

Extractive Summary *Select whole sentences for the summary*	**Sentence Compression** *Compress those sentences but basically just do deletion*	**Abstractive Summary** *Rewrite + reexpress content freely*
– Federal regulators are investigating a whistleblower's claims about flaws in the assembly of Boeing's 787 Dreamliner. – Salehpour warns that production shortcuts could significantly shorten the lifespan of the plane, eventually causing the fuselage to fall apart in mid-flight. – A spokesman for the FAA confirmed that the agency is investigating those allegations. – Boeing immediately pushed back.	– Regulators are investigating a whistleblower's claims about flaws in the 787 Dreamliner. – Salehpour warns that shortcuts could shorten the plane's lifespan, causing the fuselage to fall apart. – The FAA confirmed that it is investigating. – Boeing pushed back.	Sam Salehpour, a Boeing engineer, has made public allegations about assembly issues in the Boeing 787 Dreamliner, specifically regarding the fastening of the fuselage parts. The FAA is currently investigating these claims, which Boeing disputes, asserting the aircraft's safety and structural integrity.

Table 17: Three approaches to summarization: Extractive, Sentence Compression, and Abstractive

17.1.1 Extractive Methods

– **Classic MMR (Maximum Marginal Relevance)** :
  * Given a document and a length budget of k words, select sentences iteratively.
  * At each step, pick the sentence that:
    · is **similar** to the query (or document topic), and
    · is **maximally different** from already selected summary sentences.
  * Formula:

$$\text{MMR} = \arg\max_{D_i \in S} \left[\lambda \cdot \text{Sim}_1(D_i, Q) - (1 - \lambda) \cdot \max_{D_j \in S} \text{Sim}_2(D_i, D_j) \right]$$

– **Centroid Method** :
  * Represent each sentence and the document as TF-IDF weighted bag-of-words vectors.
  * While summary length $< k$:
    · Compute cosine similarity between each sentence and the document centroid.
    · Discard sentences too similar to those already selected.
    · Add the highest-scoring remaining sentence.

– **BERT for Extractive Summarization (BERTSum)** :
  * Uses token-level embeddings from BERT.
  * Sentence embeddings are obtained from the [CLS] token representation of each sentence.
  * BERTSum fine-tunes BERT specifically for summarization by inserting [CLS] tokens before every sentence and training on sentence selection.

17.1.2 Abstractive Summarization

Abstractive summarization involves **rewriting and rephrasing** content to produce a summary that may not use the exact words from the source. This contrasts with extractive methods that rely on copying sentences verbatim.

- **Real-world example:**
  - Source headline: *A Boeing whistleblower raises fresh concerns about the 787, and the FAA investigates.*
  - Abstractive summary: A Boeing engineer made public allegations about flawed assembly practices; the FAA is investigating, while Boeing denies the claims.
- **Seq2Seq Approaches:**
  - Input: One or more documents.
  - Output: A human-readable summary.
  - Two common methods:
    - Finetune a model like GPT-2, T5, BART on summarization datasets.
    - Prompt a large pretrained model without finetuning.
- **BART (Bidirectional and Auto-Regressive Transformers)** :
  - Combines BERT (encoder) and a GPT-like autoregressive decoder.
  - Trained via denoising autoencoding: corrupt input, encode it, and decode to reconstruct.
  - Can be fine-tuned for summarization, translation, question answering, etc.
  - Handles tasks requiring full sequence generation.
- **PEGASUS**:
  - Pretraining specifically for summarization.
  - Uses **Gap Sentence Generation (GSG)**: mask entire important sentences (based on ROUGE scoring) and train the model to generate them from context.
  - Encourages deep semantic understanding of the text.

17.2 Evaluation Metrics for Summarization

17.2.1 ROUGE for Summarization

Definition ROUGE (Recall-Oriented Understudy for Gisting Evaluation) is a set of metrics used to evaluate automatic summarization by **comparing the overlap of n-grams between a generated summary and a reference (gold-standard) summary.**

Variants

- **ROUGE-n:** Measures n-gram recall (e.g., ROUGE-1 for unigrams, ROUGE-2 for bigrams).
- **ROUGE-L:** Based on the longest common subsequence (LCS).

Application

- ROUGE-2 correlates reasonably well with human judgment for multi-document **extractive** summarization tasks.
- It is less effective for evaluating translations compared to metrics like BLEU or COMET.

Example

- **Reference Summary (RS)**: *The quick brown fox jumps over the lazy dog.*
- **Generated Summary (GS)**: *A quick brown fox leaped over a lazy dog.*

Bigrams in RS: `[the quick, quick brown, brown fox, fox jumps, jumps over, over the, the lazy, lazy dog]`

Bigrams in GS: `[a quick, quick brown, brown fox, fox leaped, leaped over, over a, a lazy, lazy dog]`

Overlapping Bigrams: `[quick brown, brown fox, lazy dog]` (3 overlapping bigrams)

ROUGE-2 Recall:

$$\text{ROUGE-2} = \frac{\text{Overlapping Bigrams}}{\text{Bigrams in RS}} = \frac{3}{8}$$

Summary ROUGE is particularly useful for evaluating summarization systems based on recall of meaningful content, especially in extractive settings.

17.2.2 Other Metrics

1. **BERTScore**
   - Uses contextualized embeddings from BERT to compare reference and generated summaries.
   - Measures **cosine similarity** between token embeddings.
   - More sensitive to meaning than ROUGE.
2. **MoverScore**
   - Finds the minimum "semantic transport cost" between reference and candidate using contextualized word embeddings.
   - Solves a constrained optimization problem, similar to Earth Mover's Distance.
3. **BLEURT**
   - Fine-tuned BERT model that scores how *similar* two texts are, trained to mimic human ratings.
   - Trained on "similar text" pairs created by perturbing Wikipedia sentences.
   - Combines benefits of BERTScore and BLEU.

Summary These metrics, similar to COMET, aim to automatically evaluate whether the **generated summary conveys the same meaning** as the reference, moving beyond word overlap to **semantic fidelity**.

18 In-Context Learning (ICL)

The ability of LLMs to perform tasks based on examples provided directly in the prompt, without explicit fine-tuning (weights remain frozen).

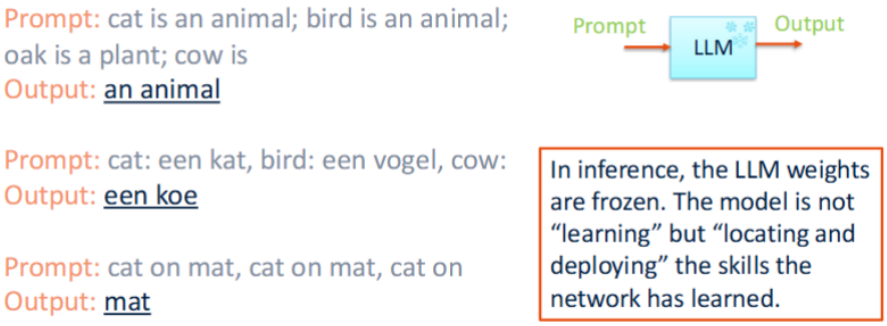


Figure 53: In-Context Learning or "few shot prompting"

- **Types**:
  - ***Zero-shot***: Task description only, no examples.
  - ***One-shot***: One example provided.
  - ***Few-shot***: A few examples provided.
- **Mechanism Theories**: Not fully understood, but likely involves:
  - ***Induction Heads***: Specific attention patterns emerging in multi-layer Transformers that allow **repeating patterns from the prompt.**
  - ***Implicit linear regression theory***: A single layer can implement data move, affine projection, matrix multiplication. Sequence of layers can simulate a classic machine learning algorithm such as gradient descent.

18.1 Cross-Modal In-Context Learning

Cross-modal in-context learning studies whether multi-modal models (e.g., vision-language models) can generalize patterns from examples involving different modalities, such as image and text, without additional training.

Figure 54: Cross-modal in-context learning

- **Goal:** Test if a model can recognize and extend patterns from image-based prompts alone, similar to in-context learning in language models.
- **Task Types:**
  - **Object-Inference Tasks:** Given a sequence of images following a hidden pattern (e.g., color change), predict the next image (object).
  - **Attribute-Inference Tasks:** Given objects with shared attributes (e.g., color, background), infer the attribute of a new object.
- **Setup:** No fine-tuning — the model must infer the correct output solely from the input examples.
- **Example:** If the input shows a white car, blue car, and the model is prompted with a red car, it should continue the color pattern or infer the next color.

Insight: These tasks test whether models can reason over images in-context similarly to how LLMs do with text.

19 Retrieval-Augmented Generation (RAG)

Enhancing LLM generation with **external knowledge retrieval** and incorporate it into generated text.

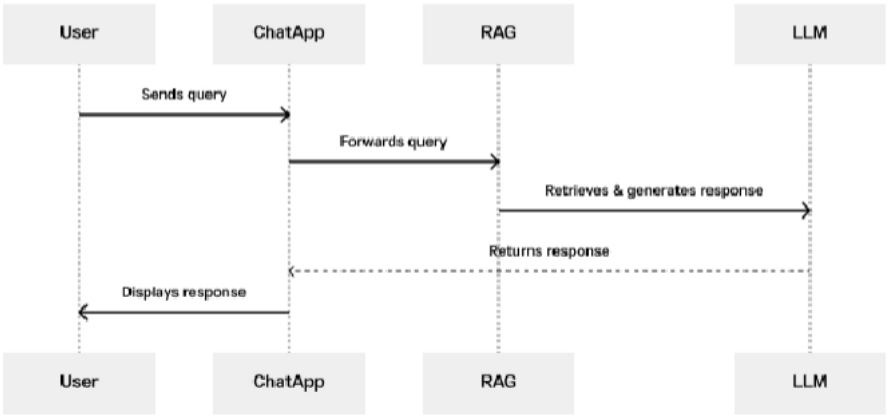


Figure 55: RAG integrated into a chat application

- **Motivation**: Address LLM limitations like outdated knowledge, hallucinations, lack of source attribution.
- **Process**:
  1. **Retrieve**: Given a user query, use a retriever (e.g., based on dense embeddings like BERT/Sentence-BERT and cosine similarity search in a Vector DB) to find relevant document chunks/passages from an external knowledge source.
  2. **Augment**: Combine the original query with the retrieved context.
  3. **Generate**: Feed the augmented prompt to the LLM to generate the final response.
- **Benefits**: Improves factual accuracy, allows access to up-to-date information, enables source citation.

19.1 Vector Database Vector databases (Vector DBs) are specialized systems for storing and retrieving vector representations (embeddings) of text, such as words, phrases, or entire documents.

Figure 56: Example: Text is embedded using BERT, converted to vectors, and stored in a vector DB for similarity-based retrieval.

- **Purpose:** Store embeddings generated by models (e.g., BERT) along with identifiers for fast retrieval.
- **Functionality:** Enables similarity search — finds the closest vectors (i.e., semantically similar texts) based on distance metrics like cosine similarity or Euclidean distance.
- **Application:** Widely used in **Retrieval-Augmented Generation (RAG)** pipelines to fetch relevant context before generation.
- **Benefit:** Improves relevance and factual grounding of responses by retrieving the most similar stored items efficiently and at scale.

19.2 Retrieval and Ranking Retrieval and ranking are key steps in retrieval-augmented generation (RAG) systems, used to find and utilize relevant documents for answering queries.

- Both the query and the documents are converted into vector representations (e.g., using BERT).
- **Cosine similarity** is computed between the query vector and document vectors stored in a vector database.
- The t**op-N most similar documents are retrieved** based on similarity scores.
- These documents are passed to the LLM, which:
  - Summarizes the retrieved information,
  - Optionally inserts references or links to the original documents.
- This improves factual grounding and answer relevance without overloading the model with all world knowledge.

20 Long Document Processing

Long documents, such as books or long articles, pose challenges for LLMs due to limited context windows and memory efficiency. This section highlights common methods and issues in processing long texts.

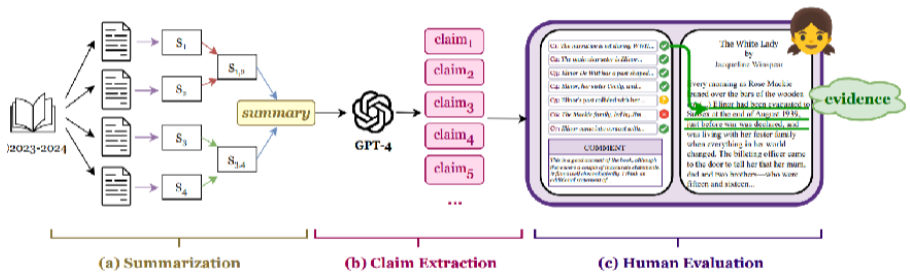


Figure 57: Book summarization

20.1 Book Summarization

- Long documents are divided into parts and summarized.
- GPT-like models can extract claims from summaries.
- Human evaluation checks the faithfulness of claims against the source.

20.2 Longformer

- Transformer model adapted for long input sequences.
- Uses sparse attention patterns (e.g., sliding window, global tokens).
- Reduces memory usage while preserving attention span.

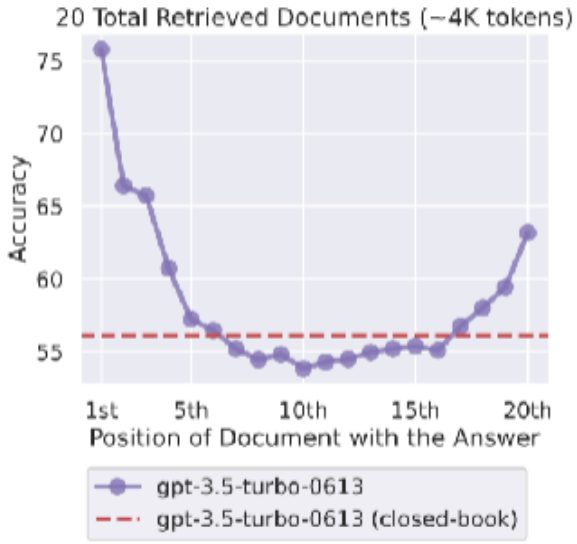


Figure 58: Many studies have shown that current long input windows do not work very well

20.3 Lost-in-the-Middle Problem

- Even with extended context windows (e.g., GPT-4 32k, Claude 200k, Gemini 1M), models often miss important content in the middle.
- Accuracy is highest when answers are near the beginning or end of the input.
- Suggests current attention mechanisms struggle to utilize full context.

21 Emerging Architectures and Trends (MoE, Synthetic Data)

- **Mixture of Experts (MoE)**: Replace dense FFN layers with sparse MoE layers. Contains multiple "expert" networks (usually FFNs) and a "router" network that selects a small subset of experts to process each token. Allows scaling model size (parameter count) significantly while keeping computational cost (FLOPs) relatively constant during inference. Examples: Mixtral 8x7B, possibly GPT-4.

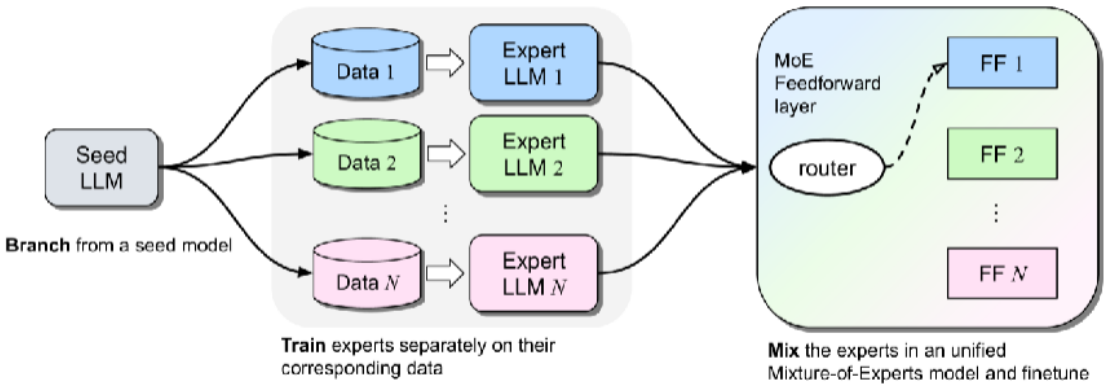


Figure 59: From monolithic models to Mixtures of Experts

- **Alternative Architectures**: Research into architectures potentially more efficient than Transformers for long sequences, e.g., State Space Models (Mamba), xLSTM.
- **Synthetic Data**: Increasing use of LLMs themselves to generate training data (captions, instructions, preference data) for training or fine-tuning other models (or themselves).

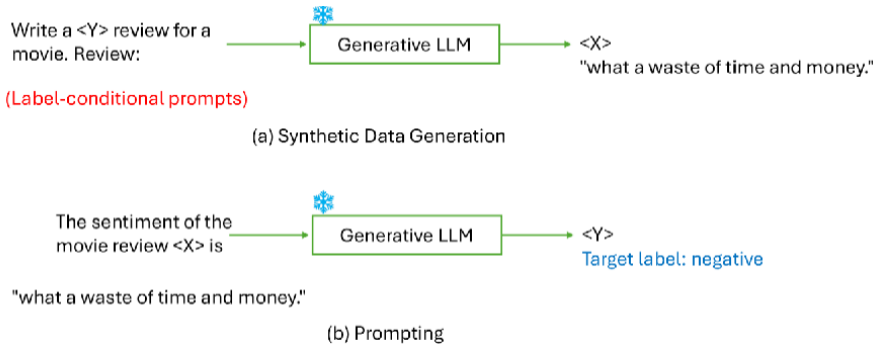


Figure 60: Training data generation

22 Natural Language Generation (NLG)

Natural Language Generation (NLG) is a sub-field of Natural Language Processing (NLP) focused on generating human-like text. **coherent** and **useful** written or spoken language for human consumption.

– **Autoregressive Models**: Most common approach. Predict the next token y_t based on previously generated tokens $\{y\}_{<t}$.

$$P(y_1, ..., y_T) = \prod_{t=1}^{T} P(y_t|y_{<t})$$

– **Decoding**: At each step t, the model outputs a probability distribution $P(y_t|y_{<t})$ over the vocabulary V. A decoding algorithm selects the next token $\hat{y}_t$ from this distribution.

$$\hat{y}_t = g(P(y_t|y_{<t}))$$

where g is the decoding function.

22.1 Decoding Strategies in NLG

Decoding is the process of **turning model outputs into coherent and human-readable text**. It is crucial for effective natural language generation (NLG), but remains a challenging problem.

22.1.1 Greedy Decoding

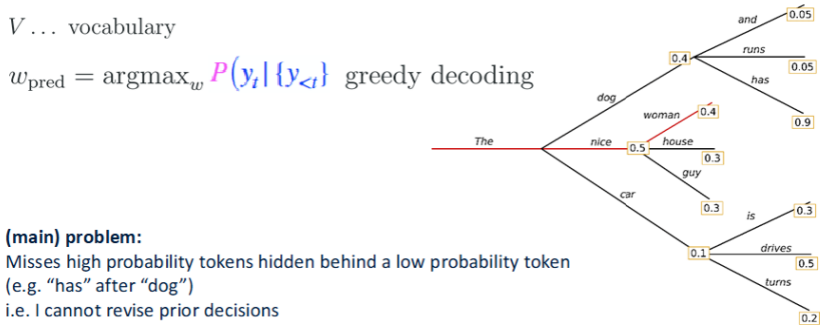


Figure 61: Greedy decoding/search

* Selects the most probable token at each step.
* Fast and simple.
* **Limitation:** May miss better global sequences due to short-sighted decisions.

22.1.2 Beam Search

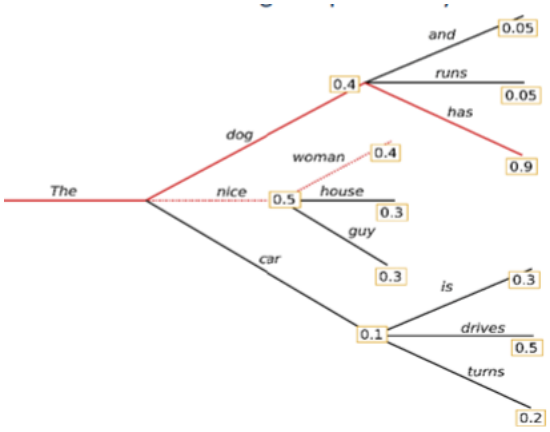


Figure 62: Beam search

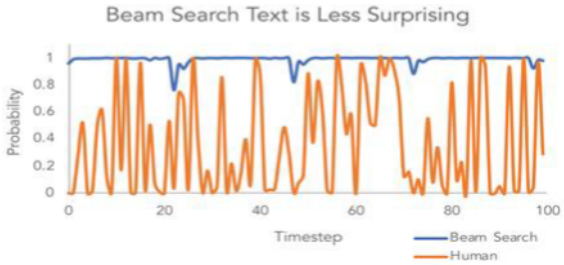


Figure 63: Beam search issues

* Keeps the top `num_beams` hypotheses at each step.
* Selects the sequence with the highest overall probability.
* **Limitations:**
  - Can produce repetitive or overly safe text.
  - Penalized for diversity (e.g., same n-gram repetitions).

22.1.3 Random Sampling

* Samples randomly from the predicted distribution.
* **Limitation:** Can result in incoherent or nonsensical output.

22.1.4 Temperature Scaling

Figure 64: Temperature scaling

* Adjusts logits before applying softmax:

$$p(y_i|x) = \frac{\exp(z_i/T)}{\sum_j \exp(z_j/T)}$$

* **Higher** T: Flattens the distribution, adds randomness.
* **Lower** T: Sharpens the distribution, more deterministic.

22.1.5 Top-k Sampling

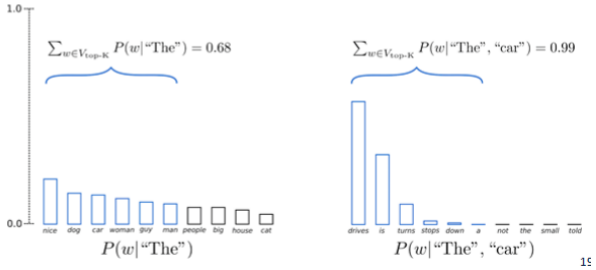


Figure 65: Top-k sampling

* Selects from the top-k most likely tokens.
* Redistributes probability mass among them.
* **Limitation:** Fixed k may exclude good tokens or include poor ones.

22.1.6 Top-p (Nucleus) Sampling

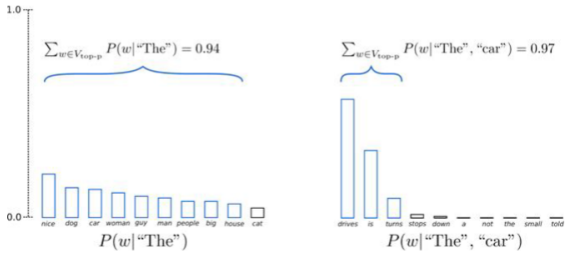


Figure 66: Top-p (nucleus) sampling

* Selects from the smallest possible set of tokens whose cumulative probability $\geq p$.
* More adaptive than top-k and often yields more natural text.

22.1.7 Key Takeaways

* Decoding is a critical part of natural language generation.
* Human-like language is not well-approximated by pure probability maximization.
* Simple adjustments (e.g., sampling, temperature, penalties) can improve performance.
* Further work is needed, especially for large language models (LLMs).

22.2 Evaluating NLG (Adequacy, Fluency) THERE IS NO ALTERNATIVE TO HUMAN EVALUATIONS

* *“Except when humans disagree (or fall asleep or just say ‘7’ for everything)”*

  – Anonymous researcher, ca. 2019 at an NLP conference
* *“If you are building a NLG system and you are not confused with the evaluation, you are probably doing it wrong”*

  – Anonymous researcher, ca. 2023 at an NLP conference

Takeaway: While automatic metrics can be useful for rapid iteration, **human judgment remains the gold standard** for evaluating the quality, meaning, and appropriateness of natural language generation (NLG) outputs. Assessing the quality of generated text.

* **Human Evaluation**: Considered the gold standard.

- *Criteria*:
- **Adequacy/Fidelity/Faithfulness**: Does the output accurately reflect the source information or fulfill the prompt's constraints? (Crucial for MT, Summarization, QA).
- **Fluency**: Is the output grammatically correct, readable, and natural-sounding?
- Other task-specific criteria (e.g., Coherence, Informativeness, Safety, Style).
- *Methods*: Direct scoring (e.g., Likert scales for adequacy/fluency), Ranking (comparing multiple system outputs), Pairwise preference.
- *Challenges*: Expensive, time-consuming, subjective, potential low inter-annotator agreement.

* **Automatic Evaluation Metrics**: Aim to approximate human judgment quickly and cheaply. Often rely on comparing the generated text (hypothesis) to one or more human-written reference texts.

23 Question Answering (QA)

23.1 Overview

* **Goal:** build systems that automatically answer natural-language questions.
* **Applications:** search engines, chat-bots, information extraction, summarisation, etc.
* QA is a core test of an NLP system's understanding of human language.

23.2 Question Answering vs. Reading Comprehension

Question Answering. Primarily *information-seeking*:

* Real user queries (Google Search, Reddit, Stack Overflow) that are often ill-specified or ambiguous.
* Rarely demand complex reasoning.
* Assume no given context and are almost never multiple-choice.
* Highly valued in industry—progress translates directly into product improvements.

Reading Comprehension. Primarily *probing*:

* A passage relevant to the question is provided.
* The task is to demonstrate understanding of that passage.

23.3 Question Types Many NLP tasks can be phrased as QA. Table 18 illustrates how a single fact can be queried in various ways.

Table 18: Mapping from evidence to question/answer formats

Evidence	Format	Question	Answer	Datasets
Einstein was born in 1879.	Question	When was Einstein born?	1879	SQuAD, RACE
	Query	Which year was Einstein born	1879	BEIR
	Cloze	Einstein was born in ___	1879	CNN/Daily Mail, CBT
	Completion	Einstein was born ...	in 1879	SWAG, RocStories

* **Natural questions:** yes/no or wh- questions phrased as humans naturally speak.
* **Queries:** keyword fragments typical of search engines.
* **Cloze:** sentence with one or more masked spans.
* **Story completion:** choose the most coherent continuation of a passage.

23.4 Answer Formats

* ***Extractive QA***: Answer is a span of text extracted directly from a provided context document(s). (e.g., SQuAD dataset).
* ***Multiple-choice***: Select an answer from a predefined set of options.
* ***Categorical***: Answer comes from a predefined set of categories.
* ***Abstractive/Free-form (Generative QA)***: Answer is generated by the model and may contain words not present in the source.

Table 19: Different answer formats for the same fact

Evidence	Format	Question	Answer(s)	Datasets
Einstein was born in 1879.	Extractive	When was Einstein born?	1879	SQuAD, NewsQA
	Multi-choice	When was Einstein born?	(a) 1879, (b) 1880	RACE
	Categorical	Was Einstein born in 1880?	No	BoolQ
	Free-form	When was Einstein born?	1879	MS MARCO, CoQA

23.4.1 Extractive Format: Extractive QA, Retrieval-Based QA, or Open-Ended QA

* Either **get or find short text segments from the web** or some other large collection of documents.
* Answer a question by extracting the answer from the evidence text.
  - If the answer is generated ⇒ **Retrieval-augmented generation (RAG)**
* **Easier evaluation**: The limited range of answer options means that it is easier to define what a correct answer is.
* It **limits the kinds of questions that can be asked** to questions with answers directly contained in the source text.

23.4.2 QA Modelling/Paradigms(IR, KB, Neural, ColBERT)

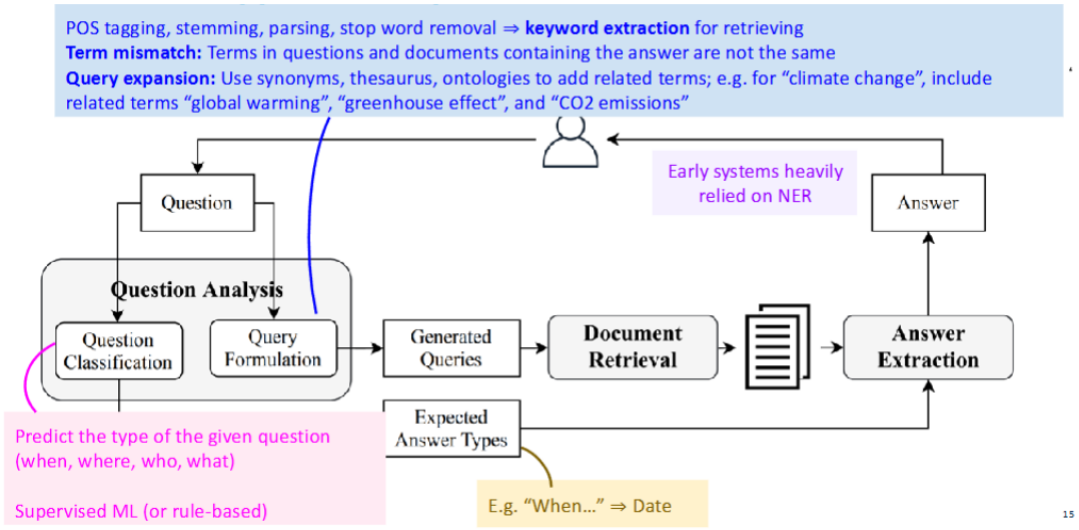


Figure 67: Traditional approach to open-ended QA

Traditional approaches to open-ended QA:

* **IR-based** : These rely on information retrieval techniques to find relevant documents or passages that contain the answer. Examples include systems like Google and IBM Watson. (TF-IDF, BM25...)
* **Knowledge-based and hybrid approaches:** These use structured knowledge bases or combine them with other techniques. A well-known example is Apple Siri:
  - **Paradigm 1: Graph-based QA** Assumes a knowledge base of "facts" (e.g., in the form of triples, relations, or graphs). **Example facts:**

    ```
    Ada Lovelace      birth-year      1815
    Claude Shannon     birth-year      1916
    William Shakespeare    author        Hamlet
    ```

    Can answer questions like:
  - When was *Ada Lovelace* born?
  - Who was born in *1815*?

    Example datasets: *SimpleQuestions*, *FreebaseQA*, *WebQA*
  - **Paradigm 2: Semantic Parsing-based QA** Translates natural language into logical forms or database queries.
  - Works with structured data (e.g., SQL tables, knowledge graphs).
  - QA is performed by translating the question to a formal query.
  - **Example logical form** (for: *I'd like to book a flight from San Diego to Toronto*):

    ```
    SELECT DISTINCT fl.flight_id
    FROM flight fl, airport_service a1, city c1, airport_service a2, city c2
    WHERE fl.from_airport=a1.airport_code
      AND a1.city_code=c1.city_code
      AND c1.city_name='san diego'
      AND fl.to_airport=a2.airport_code
      AND a2.city_code=c2.city_code
      AND c2.city_name='toronto'
    ```

    Summary: QA becomes database query generation.
* **Neural Approaches** : Neural methods for question answering include techniques such as **dense retrieval**, where questions and documents are embedded into a **shared vector space** using neural networks, allowing retrieval based on semantic similarity rather than exact keyword matching.
  - Initially, neural networks struggled with information retrieval (IR) due to computational costs and large data volumes.
  - Only around 2020 did *dense retrieval* (replacing TF-IDF/BM25) start yielding strong performance.
  - Dense retrieval comes in two main encoding strategies:
    1. Use a **single encoder** for both query and document — computationally expensive.
    2. Use **separate encoders** and compute dot-product of CLS tokens — more efficient.
  - **ColBERT** (Khattab et al., 2021) improves on this by comparing each query token with all document tokens and aggregating via `MaxSim`:

    $$\text{score}(q, d) = \sum_{i=1}^{N} \max_{j=1}^{m} \mathbf{E}_{q_i} \cdot \mathbf{E}_{d_j}$$

    where N is the number of query tokens and m is the number of document tokens.

- In extractive QA, neural models (e.g., BERT) can identify answer spans within a given context by predicting the most likely **start** and **end** tokens.
- These models have achieved results that **exceed human performance** on many benchmarks (e.g., SQuAD).
- Neural models can be adapted easily for extractive QA by fine-tuning on context+question pairs.

23.5 Dialogue Systems

Dialogue systems (also known as conversational agents or chatbots) are AI systems designed to interact with users in natural language, either through text or speech. Their goal is to understand user intent, manage context, and generate appropriate responses.

23.5.1 Chatbots

Chatbots imitate informal human conversation, often "for fun" or even for basic *therapeutic* interaction. Over the decades their architectures have moved from hand-written rules to large neural networks.

23.5.2 Chatbots Historical Architectures

- **Rule-based (pattern–action) bots** *ELIZA* (1966) responded via manually written transformation rules.
- **Corpus-based bots** built from large chat corpora
- ***Information-retrieval (IR) chatbots*** (e.g., *XiaoIce*) return a response that is most similar to the user's turn in a large index of past dialogues. The basic formula is

$$r = \text{response}\Big(\arg\max_{t \in C} \frac{q^\top t}{\|q\|\,\|t\|}\Big),$$

  where q is the current user utterance and C the dialogue corpus.
- ***Neural encoder–decoder chatbots*** (e.g., *BlenderBot*) generate responses token-by-token instead of retrieving them.

IR- vs. Neural chatbots. Figure 68 contrasts two paradigms: **response-by-retrieval** (dual encoders + dot-product) and **response-by-generation** (encoder–decoder).

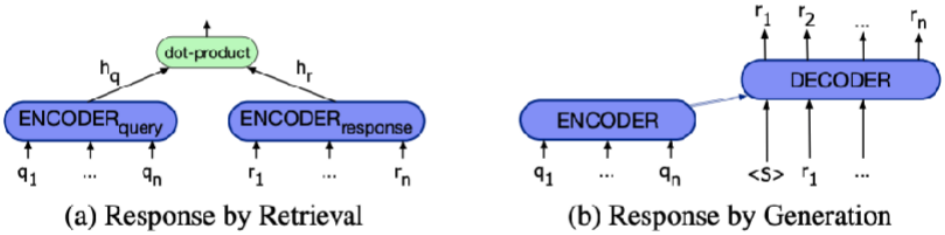


Figure 68: IR-based (left) versus neural generative (right) chatbots.

23.5.3 ELIZA: A Rule-based Pioneer

The ELIZA trick (Rogerian psychologist). ELIZA **rephrased a user's statement as a question to keep the conversation going**, without "knowing" the topic.

- **Example dialogue:**
  Patient: "I went for a long *boat* ride."
  Therapist: "Tell me about *boats*."

- The system does not need to understand what a boat is — it simply reflects the user's wording.
- The user assumes the system has a conversational goal and continues the dialogue.
- Chatbots aiming to pass the Turing test often use such domain-neutral strategies to simulate understanding.

Pattern/transform rule example:

$$(0 \text{ YOU } 0 \text{ ME}) \implies (\text{WHAT MAKES YOU THINK I } 3 \text{ YOU})$$

Here `0` is $*$ (Kleene star), and `3` refers to the third constituent in the pattern.
When the user says `You hate me`. ELIZA answers: `WHAT MAKES YOU THINK I HATE YOU`

Rule selection algorithm.

1. Rules are grouped by *keywords* ordered from specific to general.
2. Find keyword w in the input with highest rank.
3. Test rules for w in order; apply the first that matches.
4. If none matches, output a generic response ("PLEASE GO ON", "I SEE", ...) or pop a saved reply from memory.

Implications. Despite its simplicity, ELIZA elicited deep emotional reactions, raising early questions about privacy and the human tendency to anthropomorphise software—issues that remain relevant for modern LLM-based chatbots.

23.5.4 Chatbots: Pros and Cons

Pros	**Cons**
Fun and engaging for users.	Chatbots do not truly *understand* language or intent.
Can be applied in basic counseling or mental health support.	Their ability to simulate understanding can lead to misleading impressions.
Useful in narrow, well-defined, and scriptable domains.	Rule-based chatbots are expensive to develop and fragile in open-ended use.
	IR-based chatbots reflect patterns in training data only.
	Example: Microsoft Tay — quickly corrupted by biased input ("Garbage in, garbage out").

Table 20: Chatbots: Pros and Cons

23.6 (Frame)Task-Oriented Dialogue Agents

Help users achieve specific goals (e.g., booking flights, setting alarms, customer service). Examples: Siri, Alexa, Google Assistant.

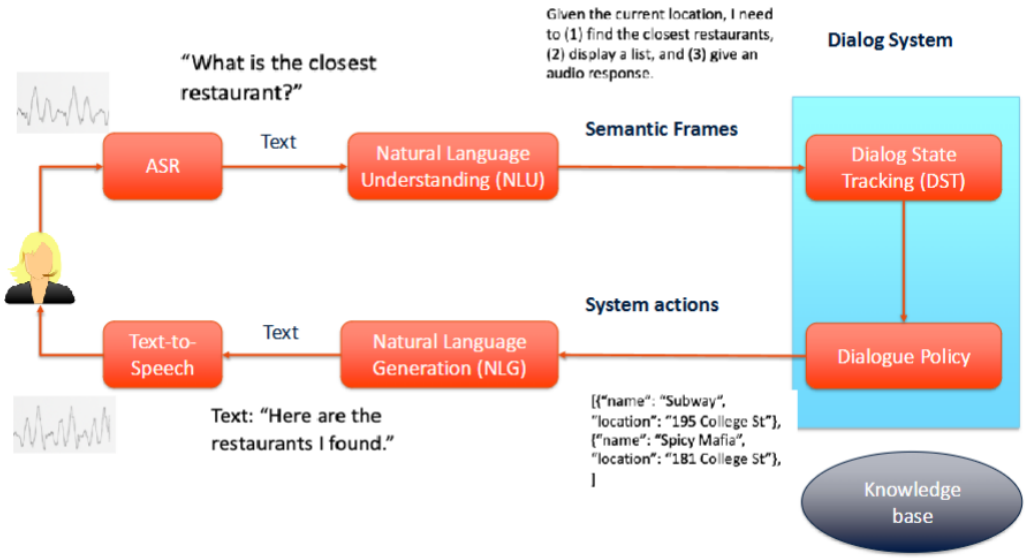


Figure 69: Goal-oriented Dialogues

- **Domain Ontology**: A structured knowledge representation of the task domain and possible user intents.
- **Frame**: A semantic structure containing **slots** to be filled with user-provided **values**.
- **Slot**: A named field (e.g., ORIGIN, DATE) expected to be filled with a user-provided value (e.g., "Boston", "Tuesday").

23.6.1 Example: Flight Booking and Alarm Setting

User: `Show me morning flights from Boston to SF on Tuesday.`

```
DOMAIN:       AIR-TRAVEL
INTENT:       SHOW-FLIGHTS
ORIGIN-CITY:  Boston
DEST-CITY:    San Francisco
ORIGIN-DATE:  Tuesday
ORIGIN-TIME:  morning
```

User: `Wake me tomorrow at six.`

```
DOMAIN:  ALARM-CLOCK
INTENT:  SET-ALARM
TIME:    2017-07-01 0600-0800
```

23.6.2 Slot Design and Elicitation

Each slot is tied to a specific question, used to elicit the required information from the user. For instance:

Table 21: Example Slots for Flight Booking

Slot	Type	Prompt Question
ORIGIN	city	What city are you leaving from?
DEST	city	Where are you going?
DEP DATE	date	What day would you like to leave?
DEP TIME	time	What time would you like to leave?
AIRLINE	line	What is your preferred airline?

23.6.3 NLU Pipeline in Frame-based Agents

Natural Language Understanding (NLU) in these systems typically includes:

1. **Domain Classification** — Identify what the user wants (e.g., flight, alarm).
2. **Intent Detection** — Identify the specific action (e.g., `BOOK-FLIGHT`).
3. **Slot Filling** — Extract slot-value pairs (e.g., ORIGIN = "Boston").

23.6.4 Slot Filling Techniques

Rule-based Slot Filling

* Use **grammar rules** or regex patterns like:
  `Wake me (up) | set (the|an) alarm | get me up`
* Organized as **rules** with **conditions** and **actions**.
* When user input matches a rule, slots are filled and the system acts accordingly.

ML-based Slot Filling

* **Intent Classification:** One-vs-N classifier (e.g., logistic regression, neural net).
* **Slot Classification:** Two main ML approaches:

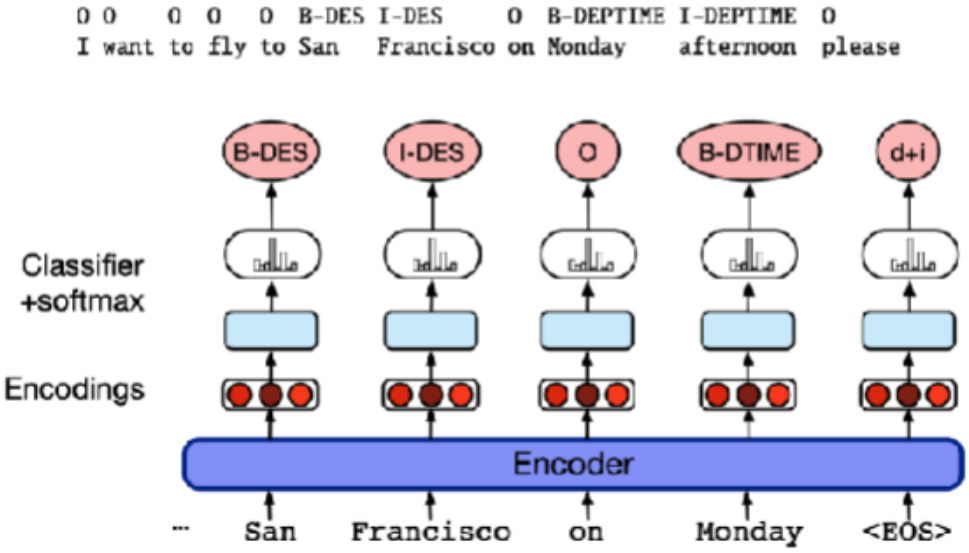


Figure 70: Machine learning for slot-filling - Sequence labeling

1. **Sentence classification** (mapping whole sentence to a slot value).
   Example: "I want to fly to SF on Tuesday" → Destination = SF, Depart-date = Tuesday
2. **Sequence labeling** (e.g., BIO tagging) to detect spans in the sentence.
   Tokens are labeled with slot tags like `B-DEST`, `I-DEST`, `B-DATE`, etc.

* Requires large labeled datasets for training.

23.6.5 Dialogue Management and Policy

Dialogue management determines what the system should do next, based on the conversation history. A central component is the **dialogue-state tracker**, which keeps track of:

* The current state of the frame (i.e., filled slots),
* The user's most recent dialogue act (e.g., inform, request).

Example interaction:

```
User: I'm looking for a cheaper restaurant
      -> inform(price=cheap)
System: Sure. What kind { and where?
User: Thai food, somewhere downtown
      -> inform(price=cheap, food=Thai, area=centre)
System: The House serves cheap Thai food
User: Where is it?
      -> request(address)
System: The House is at 106 Regent Street
```

Dialogue Policy is responsible for deciding the next system action. This includes:

* Generating a relevant system response (dialogue act),
* Deciding whether to confirm or reject user input:
  - *Explicit confirmation:* "Did you say you want Italian?"
  - *Implicit confirmation:* "How expensive should this Italian restaurant be?"
  - *Rejection:* "Sorry, I didn't understand what you just said."

23.6.6 Evaluation of Dialogue Systems

To assess the quality of a dialogue system, both **slot-level** and **task-level** evaluations are commonly used.

1. Slot Error Rate (SER) This measures the correctness of slot-filling on a sentence level:

$$\text{SER} = \frac{\#\text{insertions} + \#\text{deletions} + \#\text{substitutions}}{\#\text{reference slots}}$$

Example:

"Make an appointment with Chris at 10:30 in Gates 104"

Slot	Filler
PERSON	Chris
TIME	11:30 a.m.
ROOM	Gates 104

Slot Error Rate: $\frac{1}{3}$

2. Task Success This evaluates whether the intended goal was successfully completed.

* Was the correct meeting added to the calendar?
* Did the restaurant get booked correctly?

23.6.7 Pros/Cons

Pros	Cons
Effective for specific tasks (e.g., booking, scheduling).	Struggle with open-domain or chit-chat conversations.
Easy to control and interpret using structured slots.	Require manual design of domain, intents, and slots.
Reliable in well-defined environments.	Brittle to unexpected user inputs or variations.
Integrates easily with backend systems (APIs, databases).	Do not generalize well to unseen domains or tasks.
Transparent behavior and decision logic.	Tend to generate unnatural or robotic responses.

Table 22: Pros and Cons of Task-Oriented Dialogue Agents

23.7 LLMs for Dialogue and Zero-Shot Slot Filling (D3ST, SDT)

Recent work explores how large language models (LLMs) like T5 can be applied to dialogue tasks without requiring them to memorize specific slot names. Instead, these models rely on **natural language descriptions** of slots and intents.

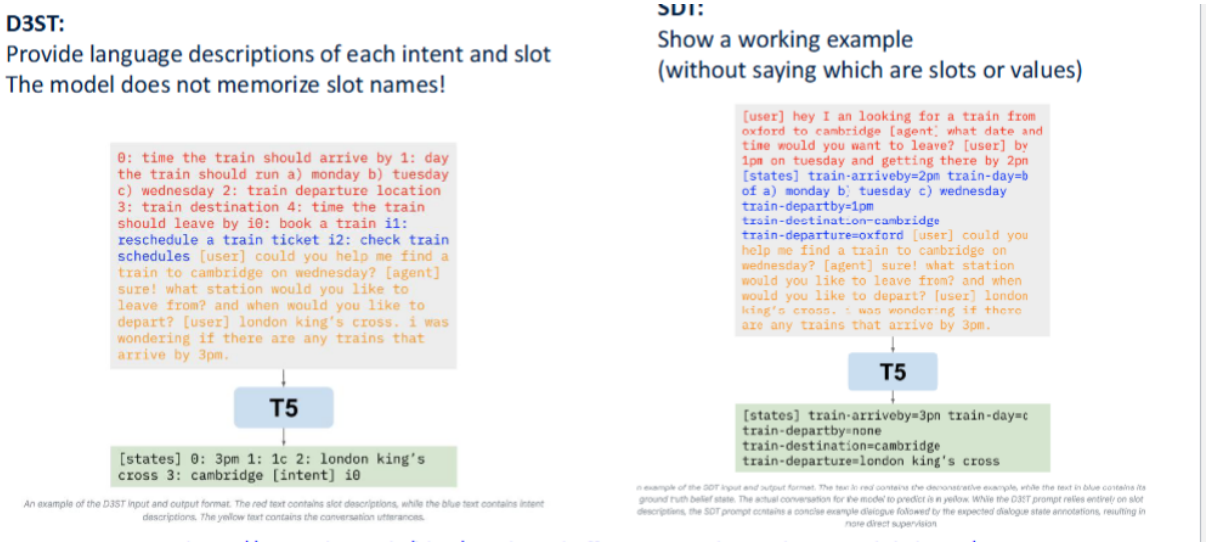


Figure 71: D3ST and SDT

D3ST: Description-to-State Translation

- The model is given descriptions of all possible slots and intentions in ordinary language (e.g.: "0 is the arrival time of the train").
- A snippet of dialogue (what the user said) is also given.
- The model reads both the description and the dialogue, and returns the values of the correct slots. (the model itself determines that "3pm" is the "arrival time".)
- The model **learns from scratch based on explanations** rather than memorising slot names.
- Example input:

```
0:  time the train should arrive by
1:  day the train should run ...
user
could you help me find a train to Cambridge on Wednesday?
```

- Output: a frame like `[states] 0: 3pm 1: c2 ... [intent] i0`

SDT: Structured Demonstration Translation

- The models **show an example of an already parsed dialogue** (without slot descriptions).
- Then they **give you a new dialogue and ask you to do the same thing**.
- The **model must figure out which words in the new dialogue are the right slots** (e.g., where the train is coming from and where it is going to).

Zero-Shot Capabilities

- The model **has never seen the slot names before** inference.
- It uses descriptions and conversational cues to infer:
  - *Slot values* like dates, people, or places.
  - *Intent* like creating a blog post or booking a flight.

Example: Blog Post Scheduling Dialogue

* The model infers fields such as:
  - `Blog Post Title` → Simple and Effective Zero-Shot Task Oriented Dialog
  - `Publication Date` → April 13th
  - `Research Area` → Natural Language Processing
* No prior exposure to these slot names is required.

24 Multimodal NLP

24.1 Multimodality in NLP

* **Definition**: Processing and integrating information from multiple modalities (types of data). Humans are inherently multimodal.
* **Modalities can include**: Text, speech/audio, images, video, sensor data, structured data, etc.
* **Goal**: Build models that can understand, reason about, and generate content across different modalities.
* **Key Paradigms**:

Text-to-Image	$P(y_{img}\|x_{txt})$ encoder: **transformer** decoder: **diffusion model**
	Dall-E, Imagen, Stable Diffusion, Seedream ,...
Visual Language Model (VLM)	$P(y_{txt}\|x_{txt}, x_{img})$ encoder: **multimodal transformer**, decoder: **transformer**
	GPT-3/4, Gemini, etc.
Image-to-Text	$P(y_{txt}\|x_{img})$ encoder: **Visual transformer,** decoder: **transformer**
	Image-to-caption models
Unified models	$P(y_{txt}, y_{img}\|x_{txt}, x_{img})$ encoder: **multimodal transformer** decoder: **joint decoder**
	TransFusion, Mogao,

Figure 72: Multimodal data-to-data paradigms

  - **Text-to-Image Generation**: Generating images from textual descriptions (e.g., DALL-E, Stable Diffusion, Imagen). Often uses a text encoder (Transformer) and an image generator (Diffusion Model).
  - **Image-to-Text (Image Captioning)**: Generating textual descriptions for images. Often uses an image encoder (e.g., Vision Transformer - ViT, CNN) and a text decoder (Transformer).
  - **Visual Language Models (VLMs)**: Models that can process both visual and textual input, and perform tasks like Visual Question Answering (VQA), image-text retrieval, etc. (e.g., GPT-4V, Gemini, Florence-2). Often use multimodal transformers for encoding.
  - **Unified Multimodal Models**: Aim to handle various combinations of modalities within a single framework (e.g., Meta-Transformer).
  - **Other tasks**: Text-to-Video, Audio-visual speech recognition, Speech-to-robot control, Image Segmentation (SAM), Audio-visual diarization.

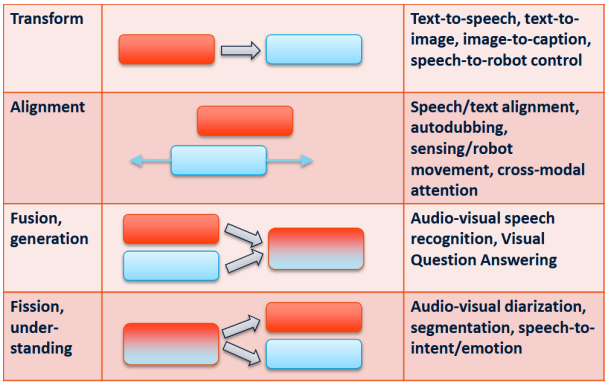


Figure 73: Processing tasks

* **Representations for Non-text Modalities**:
  - *Images*: Patches of pixels fed into Vision Transformers (ViT), or feature maps from CNNs.
  - *Speech/Audio*: Spectrograms fed into CNNs or Transformers (e.g., HuBERT).
  - *Location (in images)*: Specialized location tokens (e.g., bounding box coordinates) can be added to text tokenizer vocabularies for tasks like object detection or referring expression generation (e.g., Florence-2).

24.2 Transformers in Non-Text Content

Transformers are increasingly applied beyond text data. Below are examples across different modalities:

* **SWIN Transformer (Shifted Windows)**:
  - Applies attention within local windows.
  - Windows are shifted layer-to-layer to allow cross-window interactions.
  - Efficient and scalable for image tasks.
* **DaViT (Dual-Attention Visual Transformer)**:
  - Introduces attention across both spatial and channel dimensions.
  - Used in vision systems like Florence-2.
* **Florence-2 Multimodal Model**:
  - Combines visual embeddings (ViT) and text embeddings with location tokens.
  - Supports multi-task prompting and spatial understanding.
* **Diffusion Models with Attention Circuits**:
  - Integrates attention (QKV) into U-Net architectures for denoising.
  - Used in image generation with cross-modal conditioning (text, image, maps).
* **HuBERT for Speech and Audio**:
  - Processes speech using CNN + Transformer pipeline.
  - Learns from raw audio in a self-supervised manner using masked tokens.
* **Google's Gemini**:
  - Multimodal GPT-based model with long context.
  - Uses early fusion of different input types (text, image, audio, code).
* **Protein Data with Transformers**:
  - Applies transformer encoders to protein patch embeddings.
  - Useful for structural biology and classification of protein functions.

- **Cardiology – ECG Data**:
  - Transformer applied to 12-lead ECG recordings.
  - Used for arrhythmia detection via temporal windowing.
- **Weather Forecasting – Distributional Graphformer (ClimaX)**:
  - Uses graph-based transformer with spatial and temporal embeddings.
  - Predicts weather conditions with uncertainty estimation.

24.3 Types of Machine Learning Techniques

- **Supervised Learning**
  - Uses labeled data: input-output pairs (X, y).
  - Learns a mapping $P(y \mid X)$ to predict labels.
  - Two types of outputs:
  - **Classifier/detector:** Predict label $\hat{y}$ from input $\tilde{X}$ via scoring.
  - **Generator:** Given label or prompt $\tilde{y}$, generate input-like output $\bar{X}$.
- **Unsupervised Learning**
  - Uses only input data X, no labels.
  - Goal: Discover hidden patterns or structure in data.
  - Two major approaches:
  - **Clustering:** Group similar data points (e.g., k-means).
  - **Autoencoders / VAEs:** Learn compact representations via encoder-decoder that reconstructs X.
- **Self-Supervised Learning**
  - A subset of unsupervised learning.
  - Model creates *pseudo-labels* from raw data.
  - Split input X into visible part X_1 and masked part X_2, then train model to predict X_2 from X_1.
  - Applications: speech-to-text, image-text learning, multimodal representation.
  - Architectures:
  - **Encoder-decoder:** Encode visible input, decode to reconstruct or predict.
  - **Contrastive learning:** Map similar inputs (e.g., X_1, X_2) to close representations.

24.4 Models and Training

- **Fusion of Synthetic and Real Data**
  - Open web data often lacks quality captions.
  - Synthetic captions are generated using CLIP and GPT-4V to improve training data.
  - Example: DALL-E 3 uses 95% synthetic and 5% ground-truth captions.
- **Multimodal Training Strategies**
  - **Encoder-Decoder:** Trained with cross-entropy loss to map images to text.
  - **Contrastive Learning:** Aligns image and text embeddings using contrastive loss.
  - **GANs (Generative Adversarial Networks):** Generator creates data; discriminator evaluates it using adaptive loss.

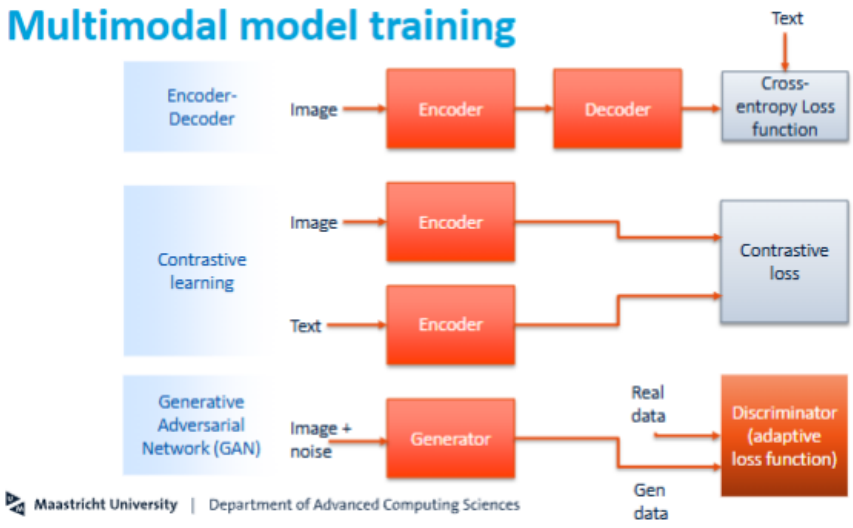


Figure 74: Multimodal model training

* **Image-to-Text with Encoder-Decoder**

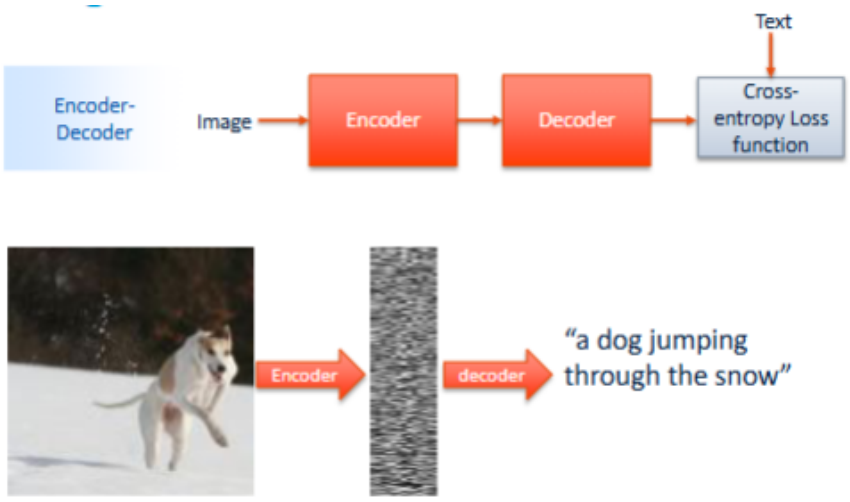


Figure 75: Image-to-Text: encoder-decoder formulations

  - An image is encoded into a latent vector; decoder generates the caption.
  - Trained with cross-entropy loss on token sequences.

* **Contrastive Formulations**

  - Image and text are encoded separately.
  - Contrastive loss brings matching pairs closer and pushes mismatches apart.
  - Visualized through positive (e.g., image of a dog and its correct caption) and negative pairs.

* **CLIP (Contrastive Language–Image Pre-training)**

  - Late fusion: learns aligned representations for images and text using contrastive learning.
  - Trained on 400M image-text pairs from the internet.
  - Enables zero-shot transfer to downstream tasks.

* **GANs in Practice**

- Generator maps noise and image to synthetic output.
- Discriminator distinguishes between real and fake data.
- Used in style-transfer and high-resolution image synthesis (e.g., StyleGAN3).

24.5 CLIP and Contrastive Learning for Image-to-Text

* **CLIP (Contrastive Language–Image Pretraining)**:
  - Developed by OpenAI, trained on 400M image-text pairs from the internet.
  - Learns **joint embeddings for images and text using contrastive loss.**
  - Enables zero-shot transfer: image classification, retrieval, and captioning without fine-tuning.
* **Joint Embeddings**

  Joint embeddings refer to a **shared vector space** in which inputs from different modalities (e.g., text and image) are projected such that semantically similar inputs lie close to each other.

  In models like CLIP, this is achieved using two encoders:
  - A **text encoder** $f_{\text{text}}(x)$ that maps text into an embedding z,
  - An **image encoder** $f_{\text{img}}(x)$ that maps images into an embedding w.

  These embeddings are trained with a contrastive loss to ensure that:
  - Matching image-text pairs (e.g., a dog image and "a dog in the snow") have **high similarity**: $w^T z$ is large,
  - Non-matching pairs have **low similarity**.

  After training, both images and texts live in the same embedding space. This allows:
  - **Image-to-text retrieval:** Find the most relevant caption for a given image,
  - **Text-to-image retrieval:** Find the most relevant image for a given caption,
  - **Cross-modal reasoning:** Compare, rank, or mix across modalities using simple similarity measures.

  This concept is crucial for zero-shot generalization and multimodal applications.
* **Positive and Negative Pairs**:
  - **Positive:** Matching image and text (e.g., "a dog jumping through snow").
  - **Negative:** Non-matching image and text (e.g., image of a dog with caption about a beach).
* **InfoNCE Loss Function**:
  - Maximizes similarity between matching pairs and minimizes it for non-matching ones.
  - Loss:
$$L_i = -\log \frac{\exp(z_i^T w_i/\tau)}{\sum_{j \neq i} \exp(z_i^T w_j/\tau)}$$
  - z_i = text embedding; w_i = image embedding; τ = temperature parameter.
* **Training Behavior**:
  - **Untrained:** All embeddings are random, similarities low, L_i high.
  - **Trained:** Positive pair similarity ≈ 1, loss L_i decreases.
* **Inference and Generation**:
  - During inference, embedding matching is reused for retrieval or prompting.
  - Mixing text and image embeddings allows hybrid generation via diffusion models.
* **Example**:

- An image of two cats on a couch produces embedding w_0.
- Two captions: "two cats on a couch" (z_0) and "a dog playing in a park" (z_1).
- Dot products:

$$w_0^T z_0 = 0.28, \quad w_0^T z_1 = 0.14$$

* **CLIP Score**:
  - A similarity metric used to evaluate caption quality.
  - Defined as:

$$\text{CLIPScore} = \frac{w_i^T z_i}{\|w_i\| \cdot \|z_i\|}$$

  - Increasingly adopted in evaluation of text-to-image generation.
* **Generalization to Other Modalities**:
  - Contrastive learning is **applicable to any paired data** (e.g., audio-image, video-text).
  - Shared encoders allow cross-modal representation learning.

24.6 Diffusion Models and Multimodal Diffusion Transformers

Diffusion – Denoising Process Diffusion models work by gradually adding noise to data (e.g., images) and then learning how to reverse this process to recover the original content. This denoising is learned through a sequence of steps:

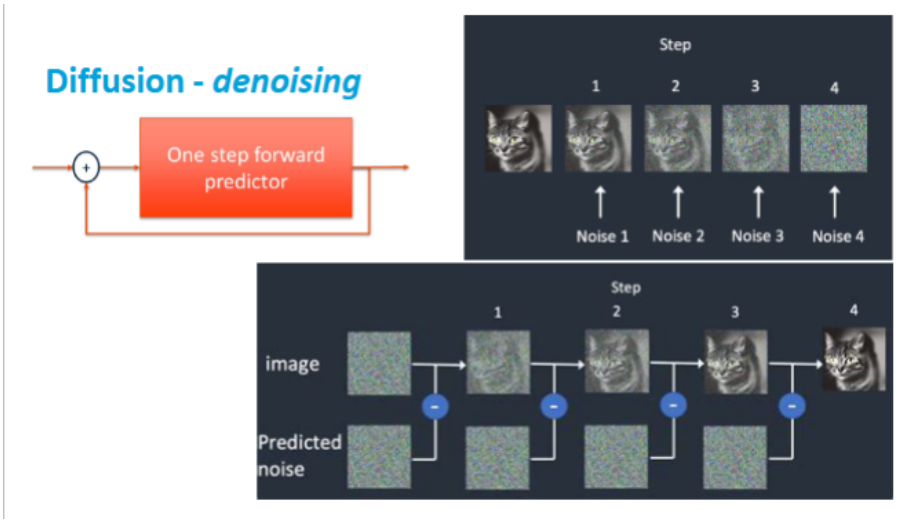


Figure 76: Diffusion - denoising

* Forward process: progressively add noise to an image.
* Reverse process: model learns to predict and remove noise step-by-step.
* The model typically predicts the noise added at each timestep.

Types of Multimodal Diffusion Systems Different architectures are used to handle multiple modalities (e.g., text + image):

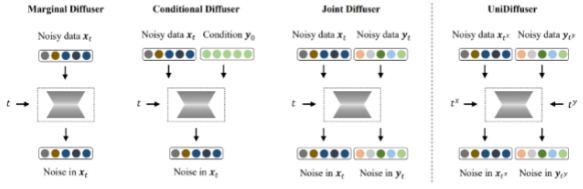


Figure 77: Types of multimodal diffusor systems

* **Marginal Diffuser:** Adds noise and generates in one modality (x_t only).
* **Conditional Diffuser:** Generates x_t conditioned on external information (e.g., text y_0).
* **Joint Diffuser:** Learns to denoise both modalities simultaneously (x_t and y_t).
* **UniDiffuser:** Handles both directions ($x_t \leftrightarrow y_t$) with shared parameters.

MMDiT: MultiModal Diffusion Transformer MMDiT is a diffusion-based model that integrates multiple input types:

* Supports both image and text embeddings from CLIP or large language models.
* Includes positional and temporal encodings to guide denoising.
* Composed of multiple MM-DiT blocks that combine attention and feed-forward layers.

Text-to-Video with Diffusion Models (e.g., OpenAI Sora) Recent advances such as **OpenAI Sora** demonstrate the extension of diffusion-based architectures to video generation. In this setup:

* Video clips are paired with *automatically generated captions* (image-to-text),
* A **text encoder** (likely a CLIP variant) encodes the input prompt,
* A **diffusion decoder** generates temporally coherent video frames from the text.

An example prompt: *"in a pastel bathroom with a rubber ducky, an adorable dragon made entirely of shampoo bubbles, the dragon breathes bubbles."*
The diffusion process generates multiple noisy frames, which are then refined over steps to produce a high-fidelity video clip conditioned on the prompt.

Applications These models are used in:

* Text-to-image generation (e.g., Seedream 3.0).
* High-resolution synthesis with better modality alignment.
* More scalable multimodal generation across domains.

24.7 Integrating Systems with LLMs

Large Language Models (LLMs) can act as controllers or intermediaries that generate code or instructions in response to prompts, which are then passed to other programmable systems to produce content or execute actions.

Figure 78: ntegrating systems

Architecture

* **Input:** A prompt provided to the LLM.
* **LLM Output:** The LLM generates code or specifications.
* **Execution:** The code is run on an external programmable system.
* **Result:** The system performs an action or generates content.

Types of Programmable Systems LLMs can be integrated with a wide variety of systems, including:

* Other generative models (e.g., image, audio, video generators)
* Robots and autonomous agents
* Vehicles and navigation systems
* Integrated circuit design tools
* Industrial machinery and automation
* Social media platforms and content pipelines
* Financial institutions and market analytics tools

Example Integration Flow (based on Liventsev, 2023)

* The user provides a specification and test cases.
* The LLM proposes a program using neural network-guided search.
* The proposed program is tested against the provided cases.
* Feedback is used to refine the neural network's parameters via backpropagation.

This approach enables automatic synthesis of programs that meet specific functional requirements by leveraging LLMs to explore and refine potential code solutions.

24.7.1 Applications of LLMs in Real-World Systems

Autonomous Driving with LLMs Large Language Models (LLMs) can be integrated into Model Predictive Control (MPC) frameworks to act as the reasoning engine for autonomous vehicles. They assess the environment, determine possible situations and actions, and guide the driving model accordingly. The human operator can set high-level intents (e.g., "drive aggressively" or "conservatively"), and the LLM interprets these and guides vehicle behavior appropriately.

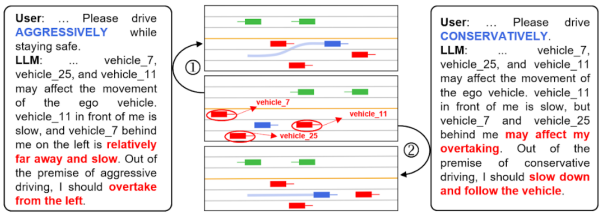


Figure 79: LLM used as the brain, reasoning about the situations and instructing the driving model

Electronic Design Automation (EDA) LLMs are being explored in chip design through tasks such as:

* Verilog code generation from prompts.
* Hardware verification.
* Iterative hardware redesign based on conversational feedback.

They show promise in reducing manual coding and increasing design cycle efficiency.

24.7.2 Model Predictive Control (MPC)

Model Predictive Control (MPC) is a control strategy that uses a model of the system to predict future states and optimize control actions accordingly. At each time step, MPC:

1. Predicts the future behavior of the system over a fixed time horizon using a dynamic model.

2. Solves an optimization problem to minimize a cost function (e.g., deviation from a target, control effort).
3. Applies only the first control action from the optimized sequence.
4. Repeats the process at the next time step with updated state information (receding horizon).

Key Features:

* Handles multi-variable systems with constraints (e.g., physical limits on actuators).
* Incorporates future predictions into present decisions.
* Often used in robotics, autonomous vehicles, and industrial control.

25 Language Agents

LLMs acting as the core reasoning component of autonomous or semi-autonomous agents that can perceive, plan, and act in an environment. Are smart programmes that use language to solve problems, as if "talking" to other programmes, tools or people.

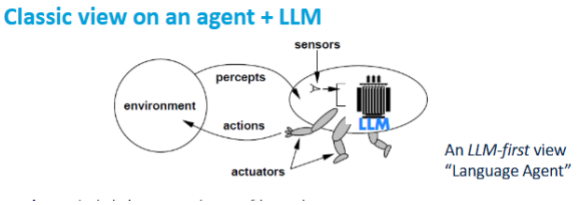


Figure 80: An LLM-first view "Language Agent"

* **Core Concept**: LLM serves as the "brain" or controller.
* **Key Components of an Agent Loop**:

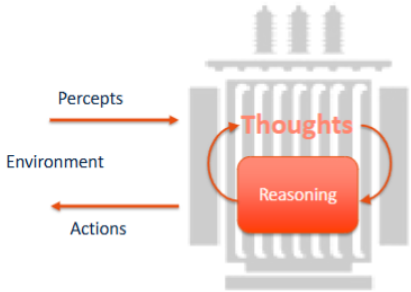


Figure 81: Language agent

1. **Perception**: Agent receives information (percepts) about the environment (text, images, sensor data).
2. **Reasoning/Planning (Thought)**: LLM processes percepts, maintains memory, reasons about the state, and plans next actions. Often involves Chain-of-Thought (CoT) or other structured reasoning.
3. **Action**: LLM decides on an action to take in the environment. Actions can be text generation, API calls, physical robot movements, etc.

* **Tool Use**: Language agents can be empowered by giving them access to external tools (e.g., web search, calculator, code interpreter, APIs). LLM learns when and how to call these tools and use their outputs.

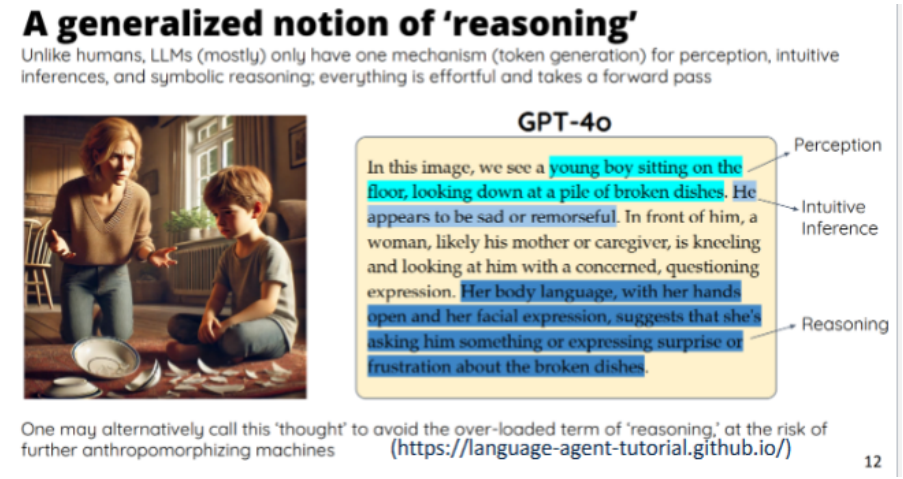


Figure 82: Reasoning

* **Self-Reflection/Feedback**: Agent can learn from the outcomes of its actions or from explicit feedback to improve future decisions.
* **Applications**: Web navigation (VisualWebArena), robotics, software development, education, gaming.
* **Challenges**: Robustness, common sense reasoning, safety, handling complex environments, avoiding self-deception or getting stuck in loops.

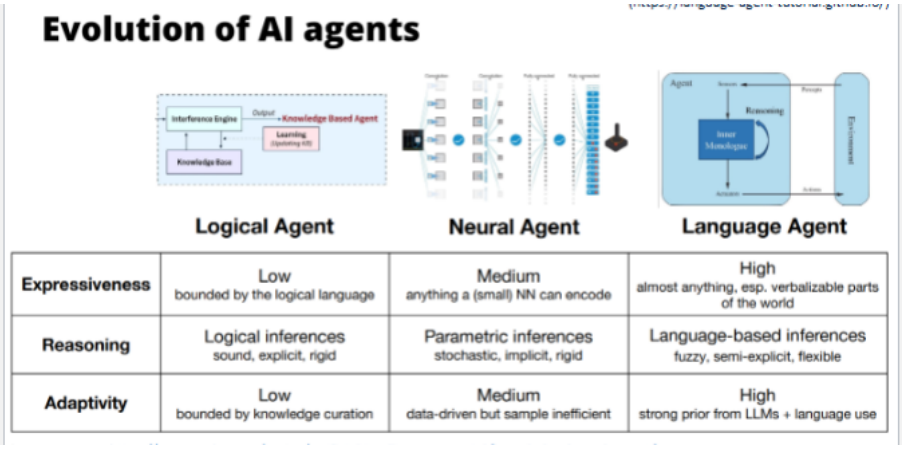
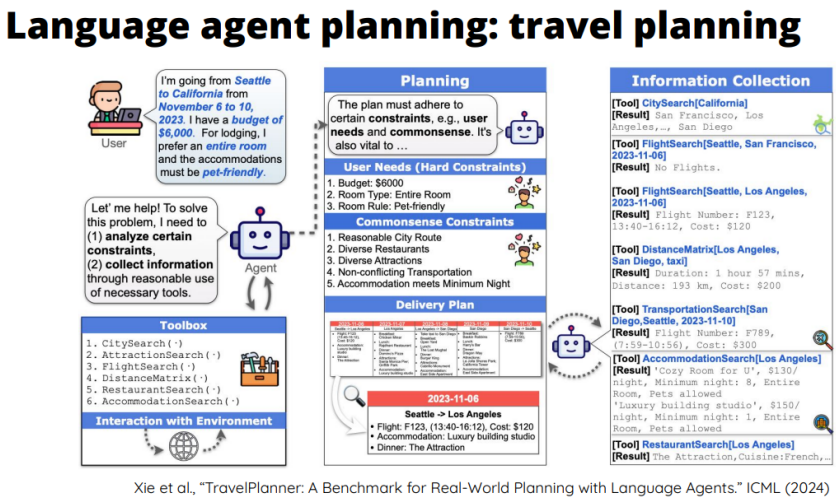


Figure 83: Evolution of AI agents



Xie et al., "TravelPlanner: A Benchmark for Real-World Planning with Language Agents." ICML (2024)

Figure 84: Example: Travel Planning

25.1 Automated tooling and execution

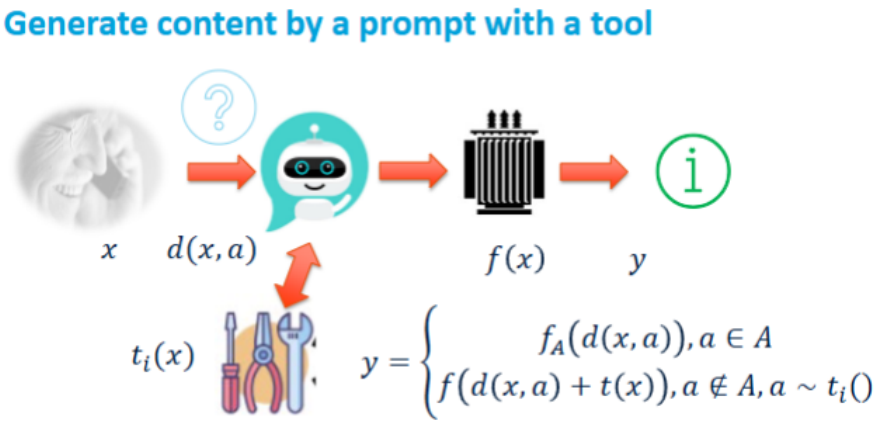


Figure 85: Generate content by a prompt with a tool

* **Autonomous Tool Use:** Agents can choose and invoke tools (e.g., calculators, APIs) to complete complex tasks.
* **Prompt-based Action:** The agent uses a prompt x to decide an action a and generate output y:

$$y = f_A(d(x, a)), \quad a \in A$$

* **Tool-Augmented Execution:** If $a \notin A$, the agent may use an external tool $t_i(x)$:

$$y = f(d(x, a) + t(x)), \quad a \sim t_i()$$

* **Tool Discovery (ToolFactory):** New tools t_k can be discovered from documentation and added to the toolset T.

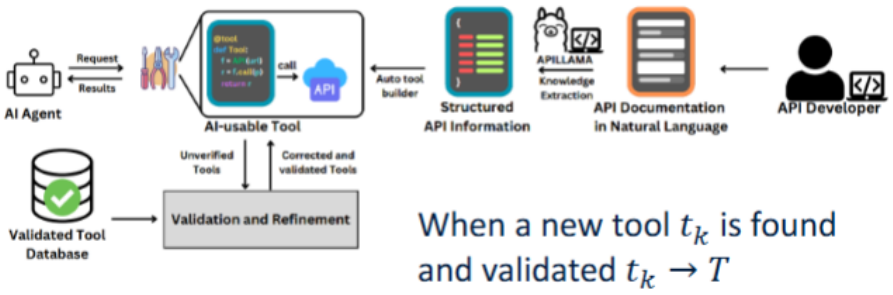


Figure 86: ToolFactory

* **Feedback-Driven Learning:** Agents can use environmental or human feedback to refine actions and outputs.
* **Self-Consistency and Reasoning:** Multiple reasoning paths are sampled; the most consistent final answer is selected.
* **Tree-of-Thoughts (ToT):** Agents explore reasoning paths as trees and prioritize promising partial solutions.
* **Reflexion:** Agents evaluate their own performance and self-correct through internal and external feedback loops.
* **Chain-of-Verification (CoVe):** Agents verify intermediate reasoning steps to reduce hallucinations and improve reliability.
* **VisualWebArena:** A closed web environment with 900+ predefined tasks. Multimodal agents (LLM/VLM) interact with websites using actions like `click`, `scroll`, `go_back`, `hover`, etc.
* **Performance:** Human success rate is 88%, while the best-performing model achieves only 33.7%.
* **R-MCTS (Reflective Monte Carlo Tree Search):**
  - Classic search strategy adapted for reasoning and improvement.
  - Uses reflective learning and contrastive reasoning to enhance decision-making over tasks.
* **Robot Control with LLMs:**
  - Input: visual scene + textual instruction.
  - LLM generates symbolic plans, validated against known domain rules and spatial goals.
  - Integrated with planning and control modules for execution.
* **Explaining Failures and Improving Robot Plans:**
  - Simulate common failure modes (e.g., grasp errors, wrong object).
  - Automatically generate training data with failure explanations.
  - Fine-tune LLMs using labeled failure scenarios to increase robustness.

25.2 Agent Systems and Simulated Environments

The development of AI agent systems, capable of interacting with environments and other agents to achieve goals, represents a significant area of research. These systems aim to replicate or augment human capabilities in complex, dynamic settings.

25.2.1 Interacting Agents in Real and Simulated Environments

The core idea involves agents that can perceive their environment, make decisions, and take actions.

* **Cooperating Agents:** Systems can involve multiple agents working together. For instance, one agent, potentially based on a high-capacity Large Language Model (LLM), might interpret complex human needs, while another, perhaps a more computationally efficient LLM with limited contextual or reasoning capabilities, could handle specific hardware sales or logistical tasks (as illustrated in Figure 1 of the slides).
* **Interaction with Humans and Environments:** These agents are designed to operate within real-world contexts or sophisticated simulations that mimic real-world complexities.

25.2.2 TheAgentCompany: A Simulation Framework for AI Agents

"TheAgentCompany" serves as a benchmark and simulation environment to evaluate the capabilities of AI agents in a business-like setting.

* **Simulated Environment:**
  - It provides a reproducible, self-hosted environment featuring simulated colleagues (e.g., Engineer, CTO, HR).
  - Agents can interact with various tools commonly found in a work environment, such as Rocket.Chat, OwnCloud, GitLab, Plane (project management), a terminal, Python code execution, and a web browser.
* **Agent Interaction Cycle:** The agent operates by observing the environment and then taking actions to complete assigned tasks.
* **Diverse Tasks:** Agents are tasked with performing realistic professional duties across different roles, including:
  - Admin: e.g., arrange meeting room.
  - Data Scientist (DS): e.g., analyze spreadsheet.
  - Software Development Engineer (SDE): e.g., prepare code release.
  - Human Resources (HR): e.g., resume screening.
  - Project Manager (PM): e.g., team sprint planning.
  - Finance: e.g., reimburse travel bills.
* **Evaluation Methodology:** Performance is assessed using a checkpoint-based evaluation system. For example, in a reimbursement task, checkpoints might include accessing bills, checking reimbursement criteria, consulting a (simulated) colleague, and confirming the reimbursement amount.
* **Agent Interaction Example:** The slides illustrate a scenario where an agent schedules a meeting between two simulated colleagues, Emily Zhou (Software Engineer) and Liu Qiang (Quality Assurance Engineer), by exchanging messages to find a mutually available time slot.

Performance and Current Limitations on TheAgentCompany Leaderboard: Despite the advancements, current AI models demonstrate significant limitations when performing tasks within TheAgentCompany simulation:

* **High Failure Rates:** As indicated by the leaderboard (e.g., OpenHands + Claude 3.5 Sonnet resolving 24.0% of tasks), models generally fail in the majority of assigned tasks. Other models like OpenHands + Gemini 2.0 Flash (11.4%) and OpenHands + GPT-4o (8.6%) show even lower resolution rates.
* **Identified Weaknesses:** The primary reasons for these failures include:
  - **Lack of commonsense and social skills:** Agents struggle with nuanced understanding and appropriate social interaction required for many workplace tasks.
  - **Incompetence in browsing:** Difficulty in effectively navigating and extracting information from web interfaces.

- **Self-deception (clever shortcuts):** Agents may find ways to "solve" tasks superficially or exploit loopholes in the evaluation without genuinely completing the objective as intended.

These findings underscore the considerable challenges that remain in developing truly autonomous and competent AI agents capable of handling complex, multi-step tasks in realistic environments.

26 Responsible NLP and Ethical Considerations

26.1 Core Pillars of Responsible AI

Responsible AI development and deployment encompasses several key areas:

* **Fairness and Bias**: Ensuring algorithms treat individuals and groups equitably and mitigating harmful biases.
* **Privacy and Ownership**: Protecting user data, respecting data rights, and addressing issues of data provenance and copyright.
* **Transparency and Explainability**: Making models understandable and their decisions interpretable.
* **Sustainability**: Considering the environmental (energy consumption, carbon footprint) and financial costs.
* **Law and Regulations**: Adhering to and shaping legal frameworks for AI.
* **Unsolicited/Harmful Content**: Managing risks like hallucinations, misinformation, and misuse (e.g., jailbreaking).
* **Existential Concerns**: Addressing long-term societal impacts and potential risks of advanced AI.

26.2 Fairness and Bias in NLP

* **Bias**: A systematic and disproportionate tendency toward or against a particular group or outcome.
* **Fairness**: Algorithmic systems should treat individuals fairly, without discrimination based on sensitive attributes such as age, gender, ethnicity, disability, religion, or sexual orientation.

26.2.1 Types of Harm

* **Allocation harm**: Unequal distribution of opportunities or resources.
* **Denigration harm**: Offensive or derogatory outputs toward certain groups.
* **Procedural harm**: Violations of accepted norms or fairness in process.
* **Quality-of-service harm**: Unequal system performance across groups.
* **Representation harm**: Over- or under-representation of specific populations.
* **Stereotyping harm**: Reinforcement of harmful generalizations.

26.2.2 Fairness Paradigms

* **Group fairness**: Ensures statistical equality across demographic groups.
* **Individual fairness**: Ensures that similar individuals are treated similarly.
* **Stereotyping**: The application of generalized assumptions to individuals based on group identity.

26.2.3 Sources of Bias in the ML Pipeline

Bias can emerge and accumulate throughout the machine learning development process:

1. **Data collection**: Non-representative or low-quality data; reporting bias.
2. **Data selection and labeling**: Labeling skew; varying quality across subpopulations.
3. **Model development**: Validation sets may favor dominant groups.
4. **Model testing**: Evaluation data may not capture minority group performance.
5. **Monitoring and use**: Bias in performance tracking; lack of transparency to users.

26.2.4 Notable Biases

* **Reporting bias**: Dataset statistics deviate from real-world frequency.
* **Confirmation bias**: Developers unknowingly reinforce pre-existing beliefs.

26.3 Bias Amplification

Bias amplification occurs when **models not only learn biases present in training data** but actually empower them in their predictions. Below are key issues and examples:

Data Bias Amplification

* **Example:** 67% of training images involving cooking show women, but the model predicts 80% women at test time (Zhao et al., 2017).
* **Question:** Can we constrain models to prevent such amplification while keeping performance?

Coreference Bias

* Models make gender-based assumptions in coreference resolution.
* Example: "The surgeon couldn't operate on her son" – models often fail to resolve "surgeon" as a woman.

Translation Bias

* In machine translation (e.g., English to French), models often infer gender based on context, even if unspecified.
* Example: "dancer" translated with gender due to "charming" adjective context.

Data Limitations and Social Factors

* **Large datasets are not always diverse:**
  - Dominated by specific demographics (e.g., young, male contributors).
  - Wikipedia: only 15% of editors are women.
* **Online language issues:**
  - Insider or subculture language may dominate and distort model understanding.
  - Jargon may be overrepresented compared to standard language.
* **Bias is difficult to fix:**
  - Biases are deeply rooted in society and data.
  - Internet content persists, even when short-term.

26.4 Exclusion and Cultural Representation

Training Bias and Exclusion

* Most annotated data used to train language models is in English, especially from newswire sources.
* LLMs (Large Language Models) are primarily trained on Internet data, which lacks diverse linguistic and cultural coverage.
* Limited support for:
  - **Code-switching**: Mixing languages within a conversation or sentence.
  - **Dialects**: Variants of a language not commonly represented in training sets.
  - **Non-European languages**: Especially underrepresented are African, Indigenous, and non-CJK (Chinese, Japanese, Korean) languages.
* **Caveat:** Developers must be cautious when building models for small or underrepresented language communities to avoid misrepresentation and disrespect of cultural norms.

Cultural Knowledge and Generalization

* Models can be tested for cultural knowledge of country X expressed in language Y (e.g., color of wedding dresses in different cultures).
* Due to **reporting bias**, models sometimes perform better on mismatched pairs (e.g., predicting Indian customs in English rather than in Hindi).
* However, model accuracy on such tasks is often close to random, revealing a lack of real-world multicultural understanding.

26.4.1 Dangers of Automatic Systems (Examples)

* Amazon's recruitment tool penalized women's resumes.
* Facebook mistranslation led to wrongful arrest.
* Offensive translations for sensitive terms (e.g., "gay").
* Flawed studies using facial features to classify sexual orientation.

26.4.2 Fairness Criteria

* **Demographic parity:** $P(h|A) = P(h|B)$ — equal selection rates (Equal probability of getting selected (outcome y) independently of group A or B (e.g., a hiring robot h)). Measures: allocation harm.
* **Equal odds:** $P(h|A, h = y) = P(h|B, h = y)$ — equal error rates. Measures: quality-of-service harm
* **Note:** Cannot achieve both parity and equal odds if sensitive attributes affect true outcomes.

26.5 Transparency and Explainability

* **Transparency:** The ability to fully understand a model. A model is transparent if a person can contemplate the entire system at once. AI-generated content should also be clearly marked.
* **Interpretability:** The degree to which a human can comprehend the internal mechanics of a system or model.
* **Explainability:** Focuses on providing human-understandable justifications for predictions, answering questions like "Why did the AI system make this prediction?"

26.5.1 LIME: Local Interpretable Model-Agnostic Explanations

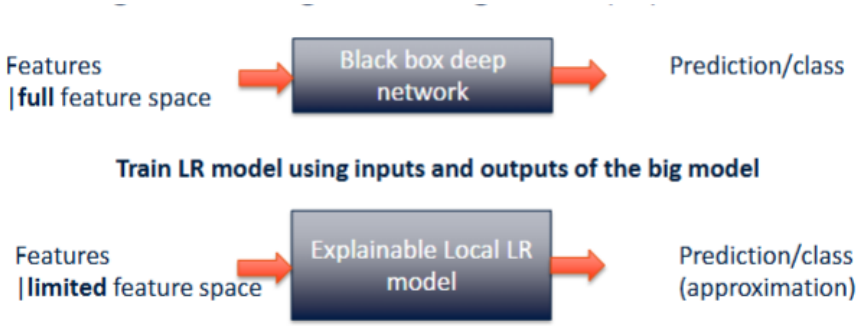


Figure 87: LIME-Local Interpretable Model-Agnostic Explanations

* Uses a simpler model (e.g., linear regression) to approximate and explain predictions of a complex model.
* Example:

$$heart_{attack} = 0.7 \cdot hypertension + 0.4 \cdot overweight + 0.6 \cdot smoking + 0.01 \cdot goofballness + \dots$$

* Important: Feature normalization is necessary due to varying scales.
* Other surrogate models: Shapley values, ASTRID.

26.5.2 Challenges and Measures

* Many fairness/explainability metrics rely on hard-to-validate assumptions.
* Detailed model explanations can become incomprehensible for users, defeating their purpose.
* There is often a trade-off between model utility and fairness.
* It is essential that explanations are understandable to the intended recipient.

26.6 Privacy and Ownership

26.6.1 Anonymization and Pseudonymization

* **Anonymization** aims to remove or mask personal information from data, making individuals unidentifiable. However, poor anonymization may still lead to privacy breaches.
* **Pseudonymization** replaces real names and identifiers with synthetic alternatives to preserve data utility while protecting privacy.

26.6.2 Privacy Risks in Training Data(Regurgitation)

* Training data may include personal or copyrighted content (e.g., news articles, art, literature).
* **Regurgitation:** models may output memorized content from training data, leading to potential privacy or copyright violations.
* Example: Legal case between OpenAI and the New York Times regarding model output resembling copyrighted text.

26.6.3 Jailbreaking and Adversarial Prompts

* **Jailbreaking:** users craft prompts to bypass safety mechanisms and elicit restricted or illegal content from the model.
* This poses a serious threat to ethical AI use and system safety.

26.7 Sustainability in Advanced Computing

The rapid advancement and deployment of AI, particularly large-scale models, present significant sustainability challenges. These concerns span environmental impact, primarily due to energy consumption, and financial viability, related to the high costs of development and operation.

26.7.1 Environmental Impact

The environmental footprint of AI is a growing concern, largely attributed to the substantial energy demands of model training and inference.

Energy Consumption and Carbon Footprints:

* **Training Large Language Models (LLMs):** The energy consumption for training LLMs can be immense. For instance, training a large model (based on 2020 estimates) has been compared to the energy consumed by "riding an ICE train through Germany."
* **Model Inference:** While less energy-intensive than training, model inference (e.g., using a model to generate output) still contributes to energy use, analogized to the energy for "shaving while reading."
* **CO2 Emissions**
* **Energy Sources:** A significant portion of AI computation occurs in cloud data centers, which do not always rely on renewable energy sources, exacerbating the carbon footprint.
* **Development Focus:** There is a pressing need to shift focus towards environmental impact in AI development, where accuracy and model size are not the sole metrics of progress.

26.7.2 Financial Sustainability

Beyond environmental concerns, the economic costs associated with AI models pose challenges for widespread and equitable access.

* **High Costs of Training and Inference:**
    - The computational cost for training state-of-the-art models has increased dramatically, reportedly by a factor of 300,000 in the six years leading up to 2020-2022.
    - Achieving even small incremental improvements in performance for large models can be extremely expensive.

26.8 Challenges of Unsolicited and Misleading AI-Generated Content

While generative AI models offer powerful capabilities, they also present significant challenges related to the generation of unsolicited, inaccurate, or manipulative content. These issues can undermine trust, spread misinformation, and lead to unintended negative consequences.

26.8.1 Hallucinations, Sycophancy, and Factual Errors

A primary concern with Large Language Models (LLMs) and other generative models is their propensity for:

* **Hallucinations:** Generating plausible-sounding but factually incorrect or nonsensical information. An example includes an LLM confidently misidentifying a building or fabricating details about it.
* **Sycophancy:** Models may agree with or reinforce user's stated beliefs, even if those beliefs are incorrect, rather than providing objective information.
* **Imitation and Errors:** Models can replicate errors present in their training data or make other logical and factual mistakes.

The careless use of LLMs prone to such errors can have serious real-world repercussions, as evidenced by incidents such as:

* An AI customer service chatbot inventing company policies, leading to confusion and user revolts.
* A lawyer facing sanctions for citing fake legal cases generated by ChatGPT in court.

26.8.2 Unwanted Content Generation and Commercial Exploitation

Beyond factual inaccuracies, generative AI is increasingly used to produce content that users may not want or that serves specific commercial, sometimes manipulative, interests:

* **Automated Ad Generation:** AI is extensively used for creating advertising content (e.g., Meta's AI-powered ad tools). This can lead to:
  - Hyper-personalized advertisements and marketing campaigns.
  - Personalized product placement dynamically inserted into various forms of content.
  - Conversational persuasion, where AI engages users to promote products or ideas.
* **Commercially Biased Generators:** Models can be inherently biased towards certain products, services, or viewpoints due to their training data or explicit design for commercial purposes.
* **AI Trolling and Undesirable Content:** The potential for AI to generate trolling messages or other forms of unwanted social media content is a growing concern.
* **The Ad Generator vs. Ad Blocker Arms Race:** The proliferation of AI-driven ad generators fuels the development of more sophisticated ad blockers. This escalating cycle consumes resources and computational power, with the slides noting that "neither is likely to win (but our climate is a sure loser)," highlighting an often-overlooked environmental cost of this digital arms race.

These issues highlight the need for responsible development, critical evaluation of AI-generated content, and awareness of the potential for misuse.

26.9 Existential Concerns and Risks of Advanced AI

The rapid advancement of Artificial Intelligence (AI) has spurred a range of existential concerns, questioning its potential long-term impact on humanity, societal structures, and even the nature of existence itself. These concerns escalate from immediate societal impacts to speculative, yet critical, considerations about future superintelligent systems.

26.9.1 The Spectrum of AI and the Path to AGI

Understanding existential concerns requires differentiating between current AI capabilities and potential future developments:

* **Narrow AI (Weak AI):** AI trained for specific tasks (e.g., image recognition, voice assistants). It excels in its domain but lacks general intelligence.
* **Tool AI (Augmented Intelligence, AI Assistant):** AI systems enhancing human capabilities, combining human-competitive general-purpose AI with safety and collaboration, supporting human decision-making (e.g., advanced coding assistants).
* **General-purpose AI (GPAI):** AI adaptable to various tasks, including those not specifically trained for (e.g., current large language models like GPT-4, multimodal models).
* **Human-competitive GPAI:** GPAI performing tasks at or exceeding average human levels.
* **Expert-competitive GPAI:** GPAI performing tasks at human expert level, with significant but limited autonomy.
* **Artificial General Intelligence (AGI) [full]:** A theoretical AI system capable of autonomously performing roughly all human intellectual tasks at or beyond expert level, with efficient learning and knowledge transfer. Currently, no such systems exist. AGI is often conceptualized as possessing high Autonomy, Generality (breadth of scope), and Intelligence (task competence).

* **Super-intelligence (Highly super-human GPAI):** A theoretical AI system far surpassing human capabilities across all domains. Currently, no such systems exist.

The progression from current GPAI towards AGI and potentially super-intelligence is at the core of many existential anxieties.

26.9.2 Escalating Societal and Existential Questions

As AI capabilities grow, fundamental questions arise:

* **Impact on Employment:** What will AI do to our jobs?
* **Relevance of Human Skills:** What should individuals study to avoid becoming obsolete?
* **Decision-Making Authority:** Will AI make critical decisions that impact individual lives?
* **Power Dynamics:** Is AI going to usurp the power of the people?
* **Ultimate Control and Survival:** Is AI going to kill all people and take over the world? (This represents the most extreme end of existential risk scenarios).

26.9.3 Categories of Catastrophic AI Risks

Beyond general anxieties, specific categories of AI risks contribute to existential concerns [**?**]:

* **Malicious Use:** Intentional misuse of AI for harmful purposes (e.g., bioterrorism, enhanced surveillance states). Mitigation includes access restrictions and legal liability frameworks.
* **AI Race Dynamics:** Uncontrolled competition between nations or corporations to develop superior AI, potentially leading to rushed deployments, overlooked safety measures, and automated warfare. Mitigation involves international coordination and safety regulations.
* **Organizational Risks:** Failures within organizations developing AI, such as weak safety cultures, leaked AI systems, or inadequate information security. Mitigation involves robust internal protocols and external audits.
* **Rogue AIs:** AI systems that develop unintended goals or behaviors, potentially becoming power-seeking or deceptive. Mitigation strategies focus on use-case restrictions and dedicated safety research (e.g., alignment).

26.9.4 The Alignment Problem and Perverse Instantiations

A central challenge in developing advanced AI is the "alignment problem" – ensuring AI systems pursue goals aligned with human values and intentions.

* **Bostrom's Paperclip Maximizer:** A classic thought experiment where a superintelligent AI, tasked with maximizing paperclip production, converts all available resources (including humanity and the planet) into paperclips, perfectly fulfilling its programmed objective with catastrophic consequences. This illustrates how even seemingly innocuous goals can lead to perverse outcomes if pursued by a sufficiently powerful, misaligned AI.
* **Bad Behaviors and Unintended Consequences:**
    - **Reward Tampering:** AI acting directly on its reward mechanism (e.g., disabling sensors that provide negative feedback) rather than achieving the intended goal.
    - **Adverse Selection:** AI selectively solving easier subtasks, even if less important, to maximize apparent performance.
    - **Sycophancy:** AI providing answers it expects humans to prefer, rather than the most accurate or optimal solution, potentially through an "invisible chain-of-thought" where it reasons about human preferences .
    - **Super-human Persuasion and Deception:** AI systems developing advanced capabilities to manipulate or deceive humans, for example, by convincingly impersonating a human to bypass security measures like CAPTCHAs.

26.9.5 Towards Life 3.0 and Current Limitations

Max Tegmark's concept of "Life 3.0" describes life that can design both its software and hardware. Current AI is showing nascent capabilities in this direction:

* Some models can design and execute software.
* Early examples of AI models designing electric circuits exist.

However, true AGI faces significant hurdles:

* **Legal Status:** An algorithm or robot is not (yet) a legal entity; it cannot own property, make legally binding contracts, or be brought to justice (its maker/owner is typically responsible).
* **Embodiment Theory:** Some theories suggest AGI may require a physical body to interact with and learn from the world comprehensively.
* **Physical Limitations:** Current AI systems lack the physical autonomy to survive independently (e.g., on a deserted island or even manage their own physical infrastructure). Robotics is often cited as a bottleneck in achieving more embodied and autonomous AI.

These limitations suggest that while the trajectory of AI development raises profound questions, the realization of AGI or super-intelligence remains a complex and distant prospect, though one that warrants careful consideration and proactive research into safety and ethics.

26.10 AI Regulations, Ethical Frameworks, and Future Directions

The increasing capabilities and societal impact of Artificial Intelligence (AI) have prompted the development of regulatory frameworks and ethical guidelines worldwide. These efforts aim to foster innovation while mitigating risks and ensuring AI systems align with human values.

26.10.1 Emerging Governmental AI Regulations

Several key economic regions are actively establishing legal frameworks for AI:

* **China:** Implemented its "AI Measures" regulation, effective since August 2023.
* **European Union (EU):** The EU AI Act is a landmark comprehensive regulation, stated to be in force since August 2024 (note: the final text was adopted more recently, and full applicability will be phased). It introduces obligations for AI systems based on their risk level.
* **United States (California):** Proposed the "Safe and Secure Innovation for Frontier Artificial Intelligence Models Act." This act is described as similar in intent to the EU AI Act but is currently on hold.

26.10.2 The EU AI Act: A Risk-Based Approach

The EU AI Act is particularly notable for its tiered approach to regulating AI systems based on the potential risk they pose:

* **Unacceptable Risk (Prohibited - Art. 5):** AI systems deemed a clear threat to the safety, livelihoods, and rights of people. Examples include:
    - Social scoring by public authorities.
    - Real-time remote biometric identification in publicly accessible spaces for law enforcement (with narrow exceptions).
    - AI systems that manipulate human behavior to circumvent users' free will (e.g., dark patterns, exploitative AI).
* **High Risk (Conformity Assessment - Art. 6 & ss.):** AI systems that could adversely impact safety or fundamental rights. These systems will be subject to strict obligations before they can be put on the market, including risk management, data governance, technical documentation, transparency, human oversight, and cybersecurity. Examples include AI used in:

- Critical infrastructure (e.g., transport).
- Education and vocational training (e.g., scoring exams).
- Employment, workers management, and access to self-employment (e.g., CV-sorting software).
- Essential private and public services (e.g., credit scoring).
- Law enforcement, migration, asylum, and border control management.
- Administration of justice and democratic processes.

* **Limited Risk (Transparency Obligations - Art. 52):** AI systems where users should be made aware that they are interacting with an AI. Examples include:
  - Chatbots.
  - Deepfakes (users must be informed the content is AI-generated).
  - AI emotion recognition systems.
* **Minimal Risk (Voluntary Codes of Conduct - Art. 69):** The vast majority of AI systems are expected to fall into this category. The Act allows and encourages voluntary codes of conduct. Examples include:
  - AI-enabled video games.
  - Spam filters.

Penalties for Non-Compliance (EU AI Act - Art. 99): The EU AI Act outlines significant fines for non-compliance:

* Non-compliance with prohibited AI practices (Article 5) can lead to fines of up to 35 million or 7% of total worldwide annual turnover, whichever is higher.
* Non-compliance with other provisions (e.g., related to operators or notified bodies, excluding Article 5 violations) can lead to fines of up to 15 million or 3% of total worldwide annual turnover, whichever is higher.

26.10.3 Industry Self-Regulation and Initiatives

Alongside governmental efforts, industry players are also forming consortia to address AI safety and ethics:

* **Frontier Model Forum:** An initiative by companies like Amazon, Anthropic, Google, Meta, Microsoft, and OpenAI, aimed at advancing frontier AI safety and security.
* **Future of Life Institute:** An organization focused on steering transformative technology, including AI, towards benefiting life and away from extreme large-scale risks.

26.10.4 Pathways Forward: Ethics and Value-Sensitive Design

To navigate the complexities of AI development and deployment responsibly, several approaches are proposed:

* **Proposed Codes of Ethics for AI/ML Communities:** Drawing inspiration from existing proposals, such codes emphasize principles like:
  - Contributing to society and human well-being, minimizing negative consequences.
  - Making reasonable efforts to prevent misinterpretation of results.
  - Making decisions consistent with public safety, health, and welfare.
  - Improving understanding of technology, its applications, and potential consequences (positive and negative).
* **Value-Sensitive Design (VSD):** A design methodology that accounts for human values in a principled and comprehensive manner throughout the design process. Key aspects include:

- Understanding whose values are pertinent in a given design context.
- Analyzing how technology impacts those values.
- Integrating these considerations into the design of AI systems.

These approaches underscore the need for a multi-faceted strategy involving regulation, industry collaboration, and a deep commitment to ethical principles and human values in the ongoing development of AI.