

2025



Lesson 01

Introduction to Generative AI (GenAI) and Large Language Models (LLMs)

Motivation for this course

- You've probably heard a lot of these GenAI buzzwords:
 - LLMs (Large Language Models)
 - Chatbots
 - RAG (Retrieval-Augmented Generation)
 - Agents
 - Function calling
 - Image/Video generation
 - Deep Fakes
 - ... more
- OpenAI's ChatGPT changed the world of knowledge forever.

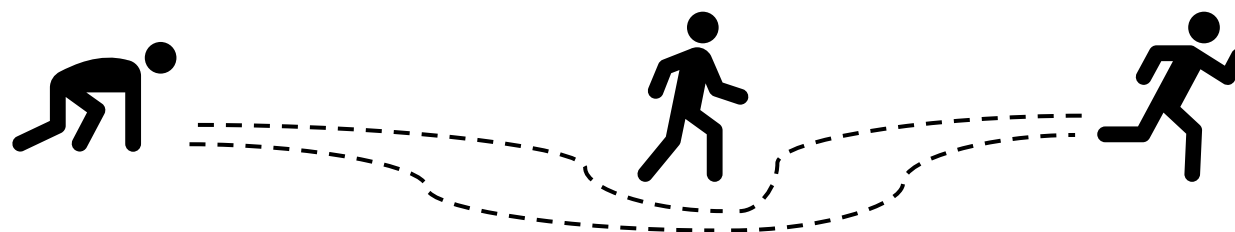


Your everyday AI companion



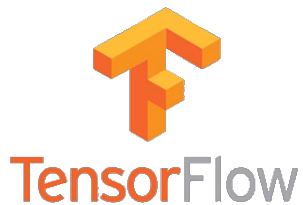
Motivation for this course

- GenAI poses a challenge for academia and practitioner.
- Development pace has far outpaced adoption.
- Familiarity with the abilities and limits of this technology is important to maintain competitiveness for us.



Tools Used in this Course

- Python: the most widely used programming language in AI/ML community.
- Tensorflow and/or PyTorch: Essential for building and training deep learning models that forms the core of Generative AI (GenAI).
- Hugging Face: A popular library for working with LLMs.
- Anaconda, Jupyter Notebook, Jupyter Lab, PyCharm, VS Code
- Platforms like Google Colab and other cloud providers supply access to cloud-based GPUs and computer infrastructure.



LangChain

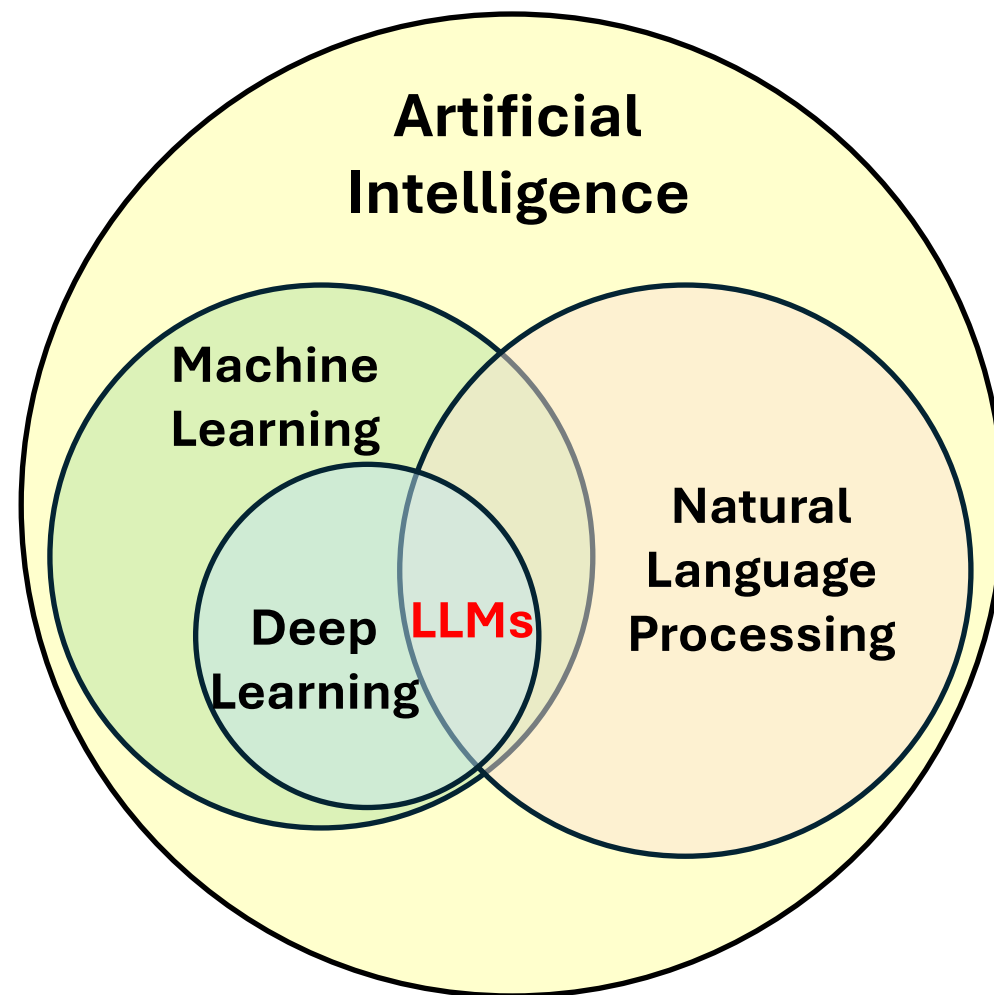


LlamaIndex

*These are just some of the tools, but there are more!

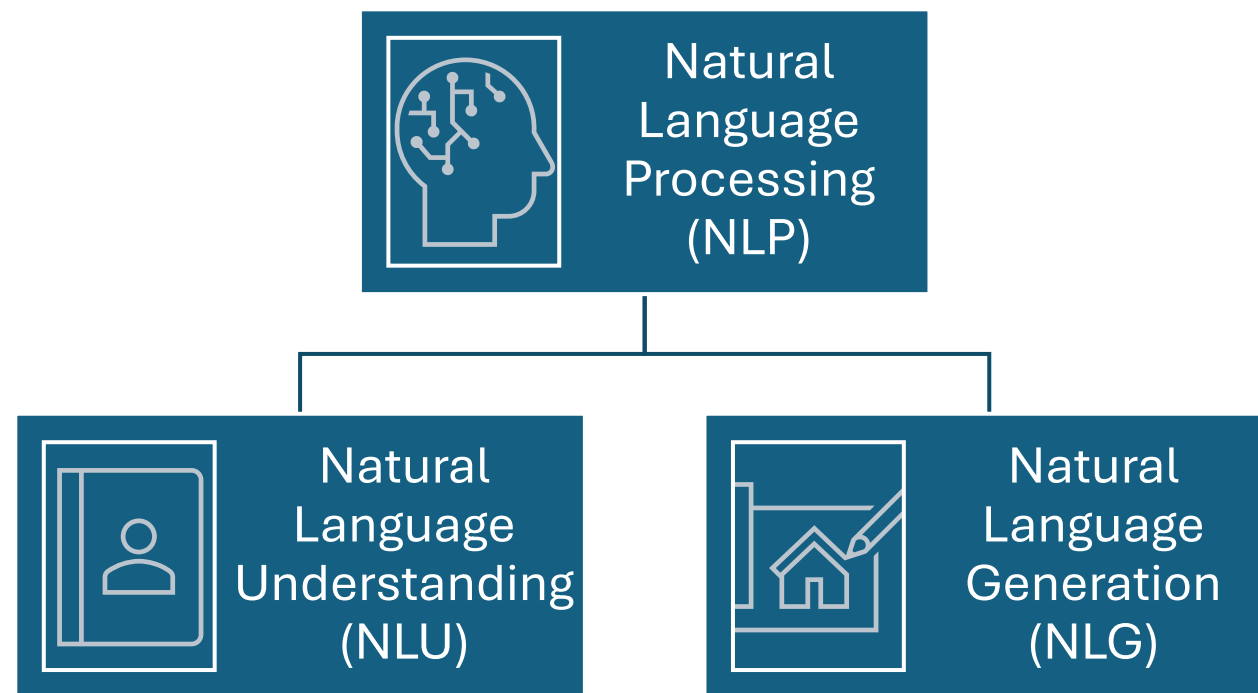
Introduction of NLP/LLMs

- Natural language processing (NLP) is a subfield of artificial intelligence and computational linguistics.
- It focuses is on enabling computers to understand, interpret and generate human language that is both meaningful and useful.
- NLP is the heart of large language models (LLMs).
- Therefore, to understand LLMs, we need to understand the concepts of NLP.



Introduction of NLP/LLMs

- NLP (Natural Language Processing) is a branch of Artificial Intelligence that uses machine learning algorithms to understand and respond in human-like language.
- NLU breakdowns unstructured user language into structured data that the computer can understand.
- NLG is the processing of producing a human language text response based on some data input.



How Does NLP Work?

- NLP involves the use of computer algorithms to analyze, understand human language or create similar language (translation).
- Many different techniques and approaches:
 - Text preprocessing – involves cleaning and preparation of text data for analysis. Tasks such as splitting text into individual words and reducing words to their basic forms.
 - Part of speech – part of speech identification i.e. nouns, verbs, adjectives.
 - Decomposition – analyzing the structure of the sentence and determine the relationships between words.
 - Semantic analysis – determine meaning of words in the context or the emotion of the text, extract entities etc.
- In short, the purpose of NLP is to enable computers to understand human language in a way that is more natural and intuitive for humans.

Tasks of NLP

- Text classification: assigning a label or category to a piece of text. For example, classifying emails as spam or not spam.
- NER (Named Entities Recognition): identifying and classifying entities like names of people, organization, location, dates and more in the text.
- Machine translation: translating text from one language to another.
- Text generation: create human-like text i.e., chatbots etc.
- Text summarization: generate a concise and coherent summary of a large corpus.
- Question & Answer: provides accurate answers to questions.
- ... more.

Basic Concepts of NLP

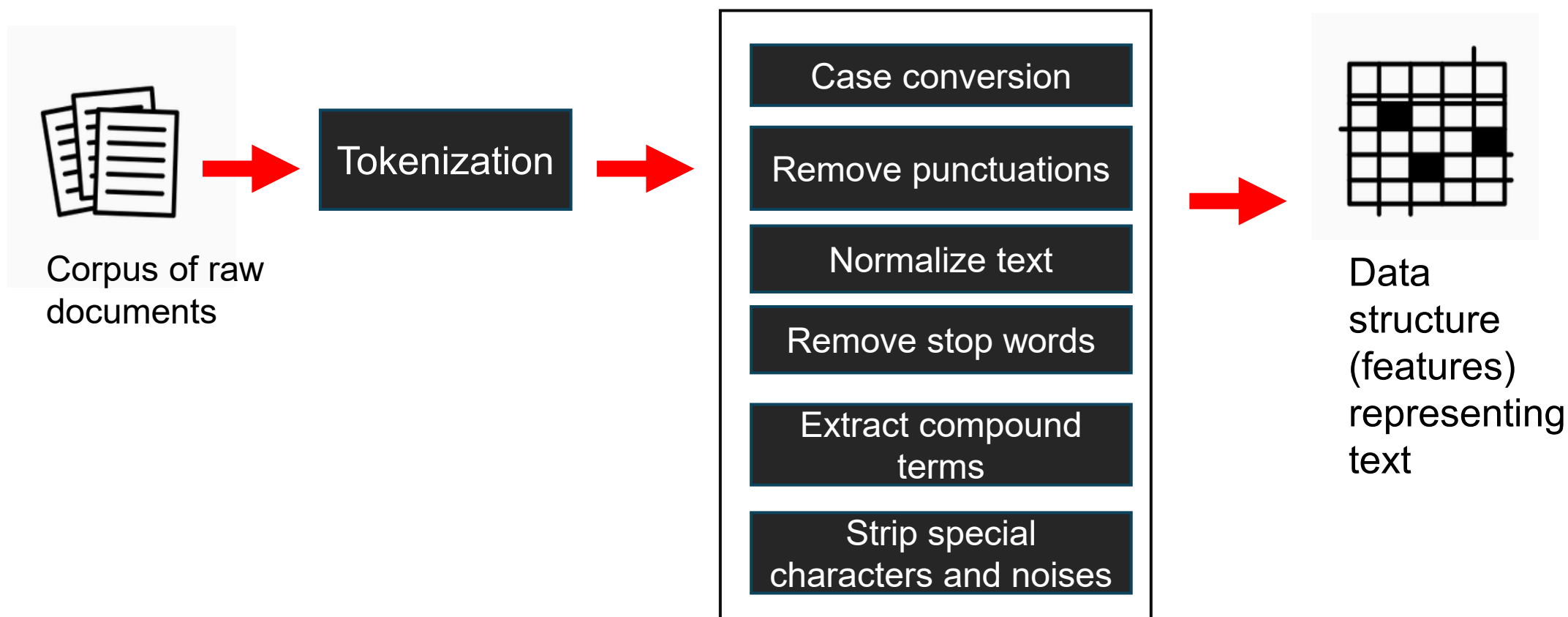
- To achieve the tasks, NLP employs a set of key concepts
- Some of the most common ones are:
 - **Tokenization**: a process to breaking down a text into smaller units i.e. the words and punctuation that it is made up of. These smaller units are called tokens and tokenization is an important preprocessing step in most NLP tasks.
 - **Stopword** removal: stopwords are common words (e.g., the, is, and) that carry little semantic meaning. Removing stopwords help to reduce noise and improve computation efficiency.
 - **Part-of-Speech** (POS) tagging: involves assigning grammatical tags (e.g., noun, verb, adjective) to each word in a sentence, indicating its syntactic role.
 - **Word embedding**: convert words to dense vector representation that capture the semantic relationships between words.

Corpus and Vocabulary

- A **corpus** refers to a large collection of text documents or utterances that are used as a dataset for language analysis and model training.
- Building and curating high-quality corpora is essential for the success of NLP application.
- A vocabulary in NLP refers to the set of unique words or tokens present in a corpus of text.
- The size of the vocabulary depends on the text corpus used to train the model.
- A very extensive vocabularies contain hundreds of thousands or even millions of unique words.

Text Preprocessing Activities

Goal: Converting unstructured text to a meaningful format for analysis.



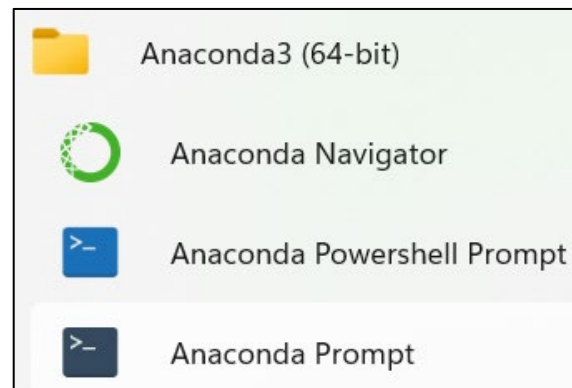
Natural Language Toolkit (NLTK)

- NLTK is a popular platform for building Python programs to work with human language data.
- It provides easy-to-use interfaces to over 50 corpora, lexical resources and trained models.
 - ➔ http://www.nltk.org/nltk_data/
- Has a suite of text processing libraries for tokenization, stemming, tagging, parsing, semantic reasoning, classifications, wrappers for other NLP libraries.
- The other popular platform is spaCy. An industrial-strength open-source library for advanced natural language processing tasks in Python.



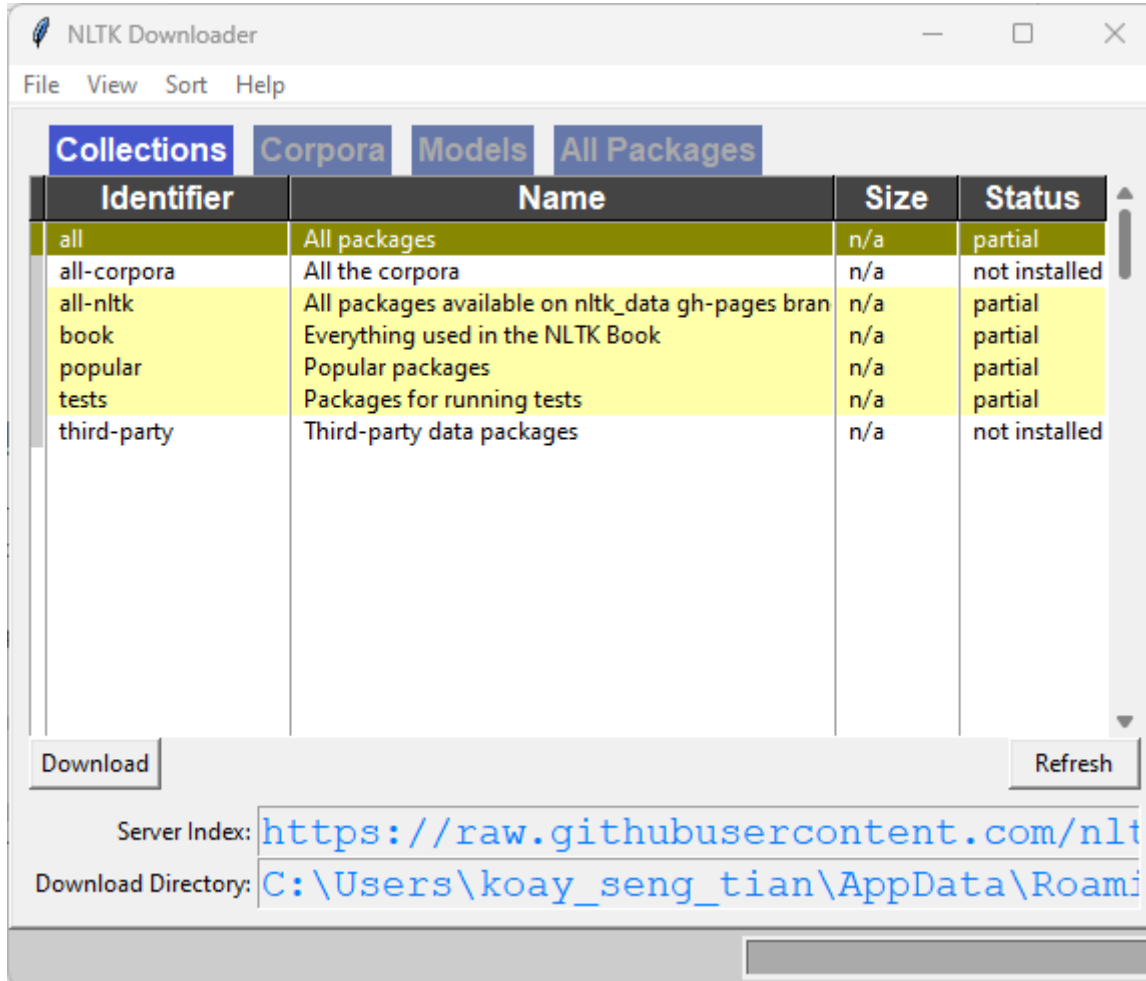
NLTK Installation

- Install Anaconda
 - ➔ Windows: <https://docs.anaconda.com/anaconda/install/windows/>
- Setup a development environment
 - Open an “Anaconda Prompt”
 - `conda create -n nlp_llm python=3.10`
 - `conda activate nlp_llm`
- Install Jupyter notebook
 - `conda install jupyter notebook`
- Install NLTK
 - `pip install nltk`
- Download all data and models from Python shell



```
>>> import nltk  
>>> nltk.download()
```

NLTK Downloads



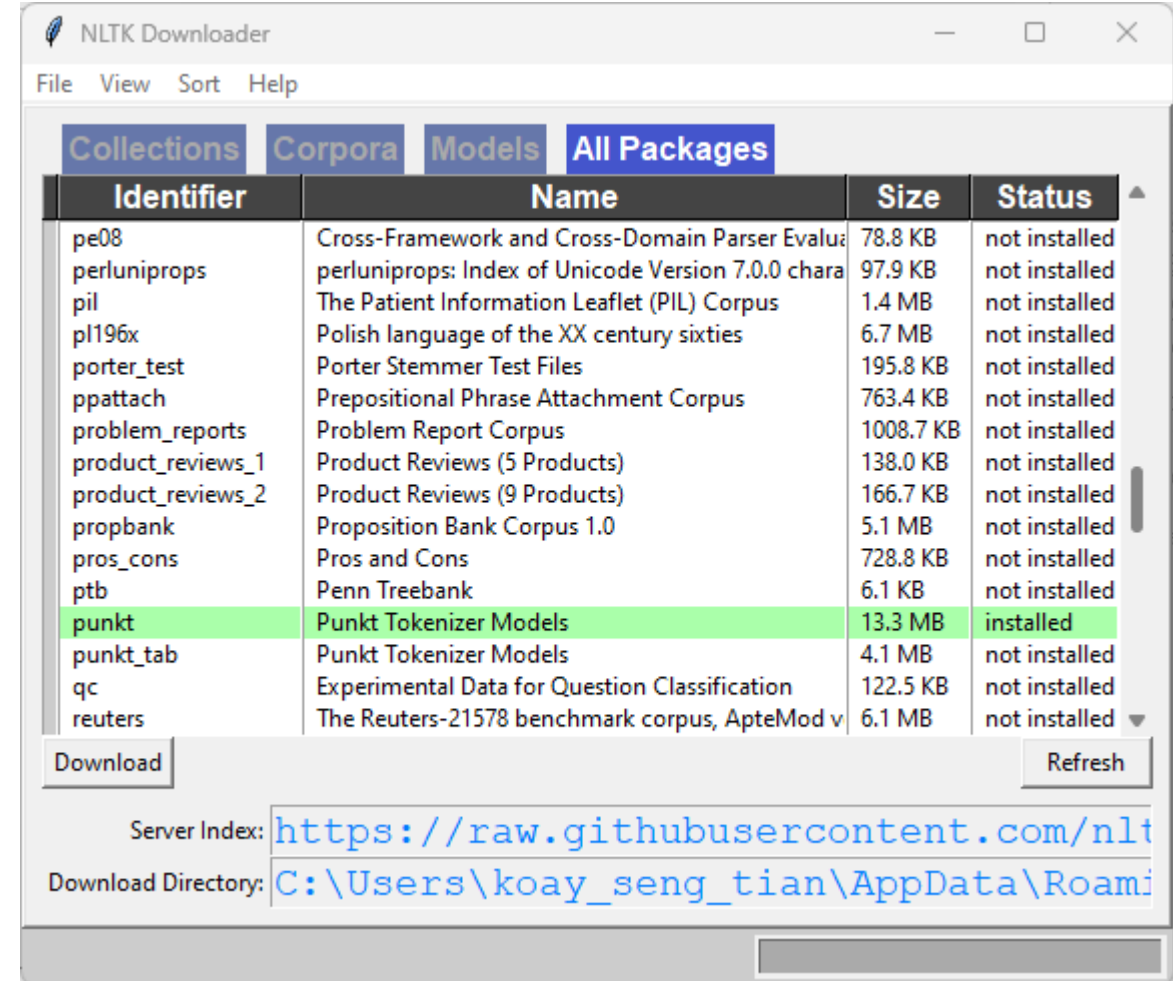
The NLTK Downloader window is shown with the 'Collections' tab selected. The table lists various collections with their identifiers, names, sizes, and installation status.

Identifier	Name	Size	Status
all	All packages	n/a	partial
all-corpora	All the corpora	n/a	not installed
all-nltk	All packages available on nltk_data gh-pages bran	n/a	partial
book	Everything used in the NLTK Book	n/a	partial
popular	Popular packages	n/a	partial
tests	Packages for running tests	n/a	partial
third-party	Third-party data packages	n/a	not installed

Server Index: https://raw.githubusercontent.com/nltk/nltk_data/gh-pages/index.html

Download Directory: C:\Users\koay_seng_tian\AppData\Roaming\NLTK

Download collections



The NLTK Downloader window is shown with the 'All Packages' tab selected. The table lists individual packages with their identifiers, names, sizes, and installation status.

Identifier	Name	Size	Status
pe08	Cross-Framework and Cross-Domain Parser Evalua	78.8 KB	not installed
perluniprops	perluniprops: Index of Unicode Version 7.0.0 chara	97.9 KB	not installed
pil	The Patient Information Leaflet (PIL) Corpus	1.4 MB	not installed
pl196x	Polish language of the XX century sixties	6.7 MB	not installed
porter_test	Porter Stemmer Test Files	195.8 KB	not installed
ppattach	Prepositional Phrase Attachment Corpus	763.4 KB	not installed
problem_reports	Problem Report Corpus	1008.7 KB	not installed
product_reviews_1	Product Reviews (5 Products)	138.0 KB	not installed
product_reviews_2	Product Reviews (9 Products)	166.7 KB	not installed
propbank	Proposition Bank Corpus 1.0	5.1 MB	not installed
pros_cons	Pros and Cons	728.8 KB	not installed
ptb	Penn Treebank	6.1 KB	not installed
punkt	Punkt Tokenizer Models	13.3 MB	installed
punkt_tab	Punkt Tokenizer Models	4.1 MB	not installed
qc	Experimental Data for Question Classification	122.5 KB	not installed
reuters	The Reuters-21578 benchmark corpus, AptMod v	6.1 MB	not installed

Server Index: https://raw.githubusercontent.com/nltk/nltk_data/gh-pages/index.html

Download Directory: C:\Users\koay_seng_tian\AppData\Roaming\NLTK

Download necessary data for tokenization

Preprocessing Techniques

1. Turn text into a meaningful format for analysis.
 - **Tokenization**
2. Clean the data.
 - Remove – capital letters, punctuation, stop words.
 - Normalization – stemming, lemmatization.
 - Chunking – linking compound terms, multi-words phrases.

Tokenization

Tokenization: takes a piece of text and converts it into small, indivisible units for text processing.

These units can be: 1) Paragraphs, 2) Sentences, 3) Words, 4) Chars, 5) N-grams etc.

The hare jumped over the dog.

Sentence

The hare jumped over the dog.

Word

The hare jumped over the dog.

Character

(The, hare), (hare, jumped), (jumped, over), (over, the), (the, dog.)

2-gram

OpenAI Token calculator: <https://platform.openai.com/tokenizer>

Code: Sentence & Word Tokenization

Sentence Tokenization

```
from nltk import sent_tokenize
```

```
text_tok = sent_tokenize("Hi Mr.Smith! I am going to buy some vegetables from the supermarket. Should I pick up some broccoli?")
```

```
print(text_tok)
```

```
['Hi Mr.Smith!', 'I am going to buy some vegetables from the supermarket.', 'Should I pick up some broccoli?']
```

Word Tokenization

```
from nltk import word_tokenize
```

```
words_tok = word_tokenize("I am learning natural language processing.")
```

```
# print the tokens
```

```
# notice punctuations like full stop is a token
```

```
print(words_tok)
```

```
['I', 'am', 'learning', 'natural', 'language', 'processing', '.']
```

Domain Specific Tokenization

- The way tokenization is done can vary greatly depending on the domain or specific task at hand.
- The reasons are:
 - **Vocabulary differences**: different domains have unique vocabularies, jargon or terminologies. Compare medical domain and legal domain.
 - **Subword tokenization**: In some domain, words might be formed from smaller meaningful units.
 - **Special character handling**: Mathematics, finance or programming use symbols and special characters and may need special handling.
 - **Language specific**: especially non-Latin script like Japanese or Chinese, requires unique tokenization requirements.
- Summary: Customizing tokenization can significantly improve the performance of NLP models in a given domain.

Preprocessing Techniques

1. Turn text into a meaningful format for analysis.
 - Tokenization
2. Clean the data.
 - Remove – capital letters, punctuation, stop words.
 - Normalization – stemming, lemmatization.
 - Chunking – linking compound terms, multi-words phrases.

Code: Remove Punctuation and Change to Lower Case

```
import string

text = """Generative AI refers to a branch of artificial intelligence that creates new content,
such as text, images, music, or even code, by learning patterns from existing data.
It uses models like GPT or GANs to generate realistic outputs, often mimicking human creativity.
This technology is transforming fields like content creation, design, and personalized experiences
by automating tasks that traditionally required human intervention."""

tokens = word_tokenize(text)

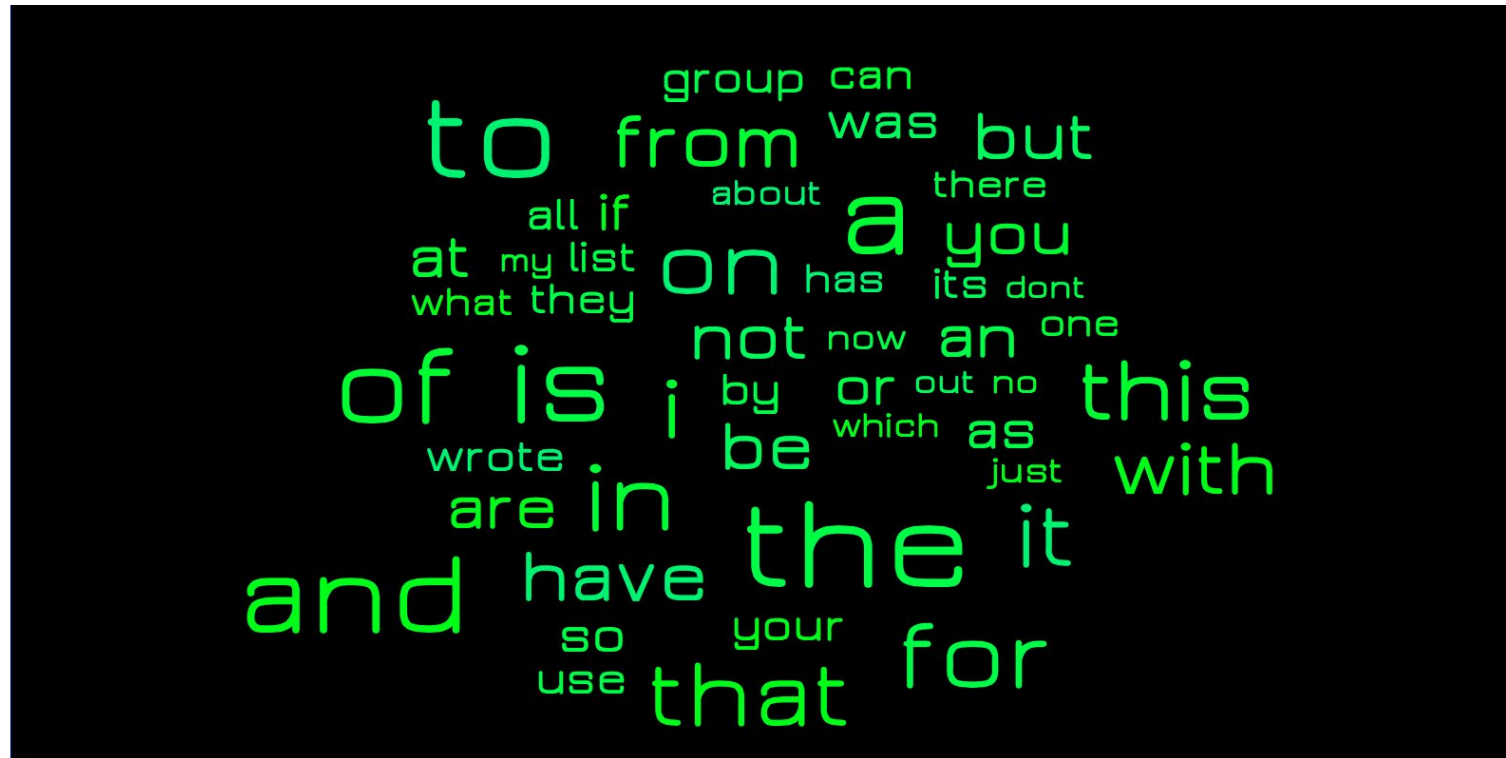
tokens_no_punctuation = []
for tok in tokens:
    if not(tok in string.punctuation):
        tokens_no_punctuation.append(tok.lower())

print(tokens_no_punctuation)
```

['generative', 'ai', 'refers', 'to', 'a', 'branch', 'of', 'artificial', 'intelligence', 'that', 'creates', 'new', 'content', 'such', 'as', 'text', 'image', 's', 'music', 'or', 'even', 'code', 'by', 'learning', 'patterns', 'from', 'existing', 'data', 'it', 'uses', 'models', 'like', 'gpt', 'or', 'gans', 'to', 'generate', 'realistic', 'outputs', 'often', 'mimicking', 'human', 'creativity', 'this', 'technology', 'is', 'transforming', 'fields', 'like', 'content', 'creation', 'design', 'and', 'personalized', 'experiences', 'by', 'automating', 'tasks', 'that', 'traditionally', 'required', 'human', 'intervention']

Stop Words

- Stop words are **hi-frequency** common words in languages that carry much less substantial information about the meaning of some text.



下面	那么
不一	那么些
不久	那么样
不仅	那些
不会	那会儿
不但	那儿
不光	那时
不单	那样
不变	那边
不只	那里
不可	那么

Text without Stop Words

The History of Humanity is part of UNESCO's General and Regional Histories Collection. The publication seeks to contribute to mutual understanding and dialogue between cultures and civilizations. This series seeks to illustrate the encounters between cultures across history and their respective contributions to the general progress of humankind. This is done through the promotion of a pluralist vision of history."

source: https://en.wikipedia.org/wiki/History_of_Humanity

Will we lose meaning and context if some words are removed?

English Stop Words

```
from nltk import word_tokenize
from nltk.corpus import stopwords

# download and install the resource
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to
[nltk_data] C:\Users\koay_seng_tian\AppData\Roaming\nltk_data...
[nltk_data] Package stopwords is already up-to-date!
```

```
True
```

```
# to print the list of stopwords in English language
stop_words = stopwords.words('english')

print(stop_words)
```

```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've", "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', 'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them', 'their', 'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these', 'those', 'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does', 'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of', 'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during', 'before', 'after', 'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further', 'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'more', 'most', 'other', 'some', 'such', 'no', 'nor', 'not', 'only', 'own', 'same', 'so', 'than', 'too', 'very', 's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll', 'm', 'o', 're', 've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "doesn't", 'hadn', "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn', "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn', "wasn't", 'weren', "weren't", 'won', "won't", 'wouldn', "wouldn't"]
```

Code: Remove Stop Words

```
sentence = "I am learning Python. It is one of the most popular programming language."
```

```
sentence_words = word_tokenize(sentence)
```

```
print(sentence_words)
```

```
['I', 'am', 'learning', 'Python', '.', 'It', 'is', 'one', 'of', 'the', 'most', 'popular', 'programming', 'language', '.']
```

```
# the code below uses python list comprehension to construct the sentence_no_stop_word list
```

```
sentence_no_stop_word = ' '.join([word for word in sentence_words if word not in stop_words])
```

```
print(sentence_no_stop_word)
```

```
I learning Python . It one popular programming language .
```

```
# the code below is the same as above code except it is not using 'python list comprehension' technique
```

```
sentence_no_stop_word = []
```

```
for word in sentence_words:
```

```
    if word in stop_words:
```

```
        pass
```

```
    else:
```

```
        sentence_no_stop_word.append(word)
```

```
print(' '.join(sentence_no_stop_word))
```

```
I learning Python . It one popular programming language .
```


Danger of Removing Stop Words

Text and labels for sentiment analysis of product reviews – before stopword removal

1. The product is really very good – POSITIVE
2. The product seems to be good – POSITIVE
3. Good product. I really liked it – POSITIVE
4. I didn't like the product – NEGATIVE
5. The product is not good - NEGATIVE

Text and labels for sentiment analysis of product reviews – after stopword removal

1. product really good – POSITIVE
2. product seems to be good – POSITIVE
3. Good product. really liked it – POSITIVE
4. like product – **POSITIVE**
5. product is good - **POSITIVE**

Preprocessing Techniques

1. Turn text into a meaningful format for analysis.
 - Tokenization
2. Clean the data.
 - Remove – capital letters, punctuation, stop words.
 - Normalization – stemming, lemmatization.
 - Chunking – linking compound terms, multi-words phrases.

Stemming

Stemming refers to a **crude heuristic process** that *chops off the ends of words* in the hope of achieving this goal correctly most of the time, and often includes the removal of derivational affixes.

Problem

- Stemmed word does not have similar meaning to the original.
- Stemmed word may not be a proper word found in the dictionary.

Lemmatization

Lemmatization is a text pre-processing technique to break a *word* down to its *root* meaning to identify similarities.

For example, a lemmatization algorithm would reduce the word *better* to its root word (or lemme), *good*.

Code: Stemming

```
stemmer = nltk.stem.PorterStemmer()
```

```
print(stemmer.stem("Production"))  
print(stemmer.stem("Products"))  
print(stemmer.stem("coming"))  
print(stemmer.stem("firing"))  
  
print(stemmer.stem("battling"))
```

```
product  
product  
come  
fire  
battl
```

Code: Lemmatization

```
nltk.download("wordnet")  
from nltk.stem.wordnet import WordNetLemmatizer
```

```
[nltk_data] Downloading package wordnet to  
[nltk_data] C:\Users\koay_seng_tian\AppData\Roaming\nltk_data...  
[nltk_data] Package wordnet is already up-to-date!
```

```
lemmatizer = WordNetLemmatizer()
```

```
print(lemmatizer.lemmatize("Production"))  
print(lemmatizer.lemmatize("Products"))  
print(lemmatizer.lemmatize("coming"))  
print(lemmatizer.lemmatize("firing"))  
  
print(lemmatizer.lemmatize("battling"))
```

```
Production  
Products  
coming  
firing  
battling
```

Preprocessing Techniques

1. Turn text into a meaningful format for analysis.
 - Tokenization
2. Clean the data.
 - Remove – capital letters, punctuation, stop words.
 - Normalization – stemming, lemmatization.
 - Chunking – linking compound terms, multi-words phrases.

Chunking

- What is the right way to tokenize open compound words?

We need to invest in **data science** and **artificial intelligence** capabilities.

"data", "science" **or** "data science"

"artificial" , "intelligence" **or** "artificial intelligence"

Mr **Heng Swee Keat** will deliver a Ministerial Statement on additional support measures for COVID-19 pandemic.

"heng", "swee" , "keat"

or "heng swee keat"

or "heng_swee_keat"

Chunking

- Chunking is the process of grouping together words into meaningful units, such as noun phrases (NP), verb phrases (VP), or prepositional phrases (PP).
- It operates on the output of part-of-speech (POS) tagging and identifies these syntactic structures to capture broader information about the text.
- For example, in the sentence "The quick brown fox jumps over the lazy dog," chunking would group "The quick brown fox" as a noun phrase and "jumps over the lazy dog" as a verb phrase.
- This helps in understanding the grammatical relationships between words and is useful for tasks like information extraction and syntactic parsing.

Code: Chunking

```
import nltk

from nltk.tokenize import MWETokenizer
from nltk.tokenize import word_tokenize

compound_words = [("artificial","intelligence"),("data","science")]
mwe_tokenizer = MWETokenizer(compound_words, separator='_')

text_1 = 'We need to invest in data science and artificial intelligence capabilities'

tokens_words_mwe = mwe_tokenizer.tokenize(word_tokenize(text_1))

print(tokens_words_mwe)

['We', 'need', 'to', 'invest', 'in', 'data_science', 'and', 'artificial_intelligence', 'capabilities']

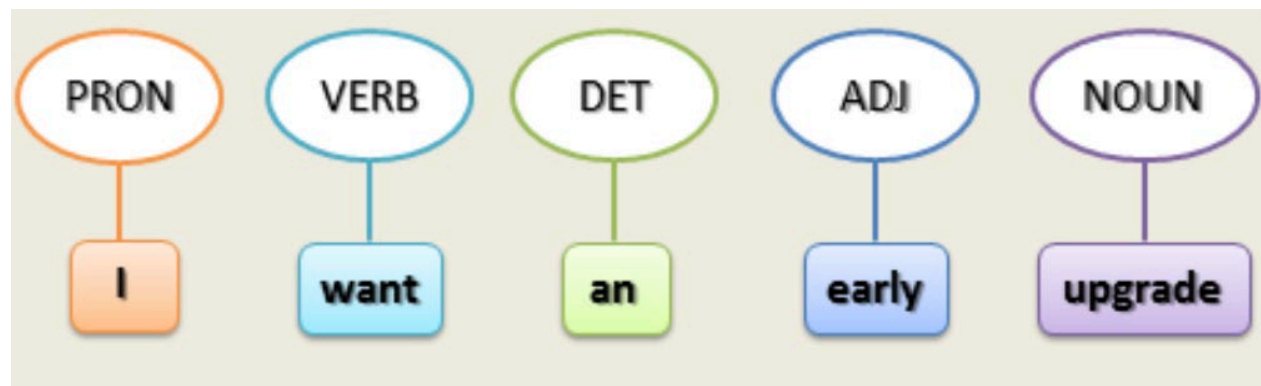
text_2 = """Mr Heng Swee Keat will deliver a Ministerial Statement on additional
support measures for COVID-19 pandemic."""

compound_words = [("Ministerial", "Statement"), ("Heng","Swee","Keat")]
mwe_tokenizer = MWETokenizer(compound_words, separator='_')
tokens_words_mwe = mwe_tokenizer.tokenize(word_tokenize(text_2))

print(tokens_words_mwe)

['Mr', 'Heng_Swee_Keat', 'will', 'deliver', 'a', 'Ministerial_Statement', 'on', 'additional', 'support', 'measures', 'for', 'COVID-19', 'pandemic', '.']
```

Part of Speech (PoS) tagging



- Sentences contain nouns, verbs, adjectives ...
- Parts of speech tagging generates labels (grammar information) for each word.
- Having tags on nouns can reveal names, entities, places etc. that are mentioned in the text.

Code: POS Tagging

```
from nltk import word_tokenize

# download and install the resource
# https://www.nltk.org/_modules/nltk/tag/perceptron.html
nltk.download('averaged_perceptron_tagger_eng')

[nltk_data] Downloading package averaged_perceptron_tagger_eng to
[nltk_data] C:\Users\koay_seng_tian\AppData\Roaming\nltk_data...
[nltk_data] Package averaged_perceptron_tagger_eng is already up-to-
[nltk_data] date!

True

# tokenize the sentence and print the tokens
words = word_tokenize("I am learning natural language processing.")
print(words)

['I', 'am', 'learning', 'natural', 'language', 'processing', '.']

# print the PoS tags
nltk.pos_tag(words)

[('I', 'PRP'),
 ('am', 'VBP'),
 ('learning', 'VBG'),
 ('natural', 'JJ'),
 ('language', 'NN'),
 ('processing', 'NN'),
 ('.', '.')]

```

Exercise

Given the text below, what are the preprocessing techniques you could apply?

"Generative AI is revolutionizing many industries by automating creative tasks such as writing, designing, and even composing music. These models, like GPT and DALL·E, learn from vast amounts of data to produce original content that mimics human output. However, the ethical implications of this technology, including concerns about bias and intellectual property, are raising important debates. As generative AI continues to evolve, it is crucial for researchers and developers to address these challenges to ensure responsible and fair use."



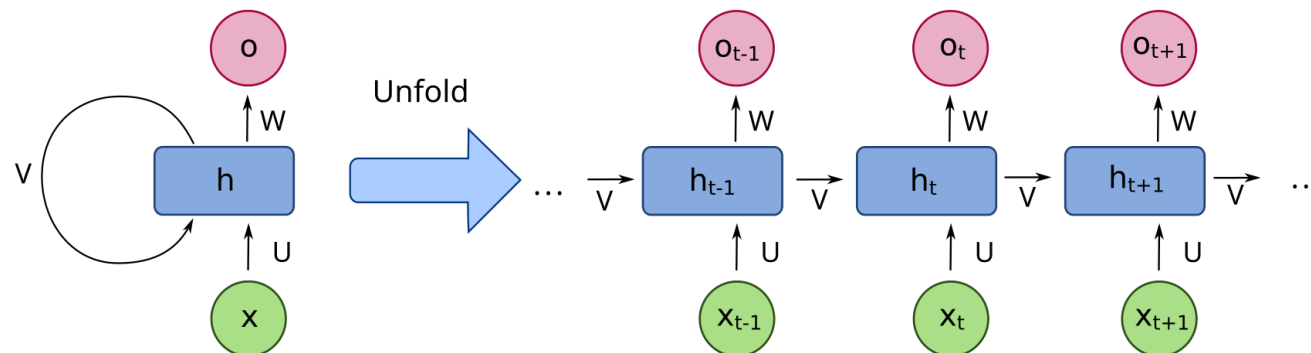
Transformer

Language Models

- Language model is designed to predict the likelihood of a sequence of words occurring in a language.
- These models learn the statistical properties and patterns present in a given language to generate text.
- Language models can broadly classify into two categories:
 - **Generative language models**: designed to generate new text based on the patterns learnt from the training data.
 - The model takes a prompt or starting sequence and then generate the next word or sequence of words one step at a time.
 - Example: GPT models (e.g., GPT-4): Generate text given a prompt.
 - **Predictive language models**: predict the likelihood of the next word in a given context.
 - Widely used in tasks like autocomplete, next-word prediction and machine translation.
 - Example: Transformers (e.g., BERT, RoBERTa): Pre-trained models fine-tuned for tasks like sentiment classification or question answering.

RNN (Before Transformer ...)

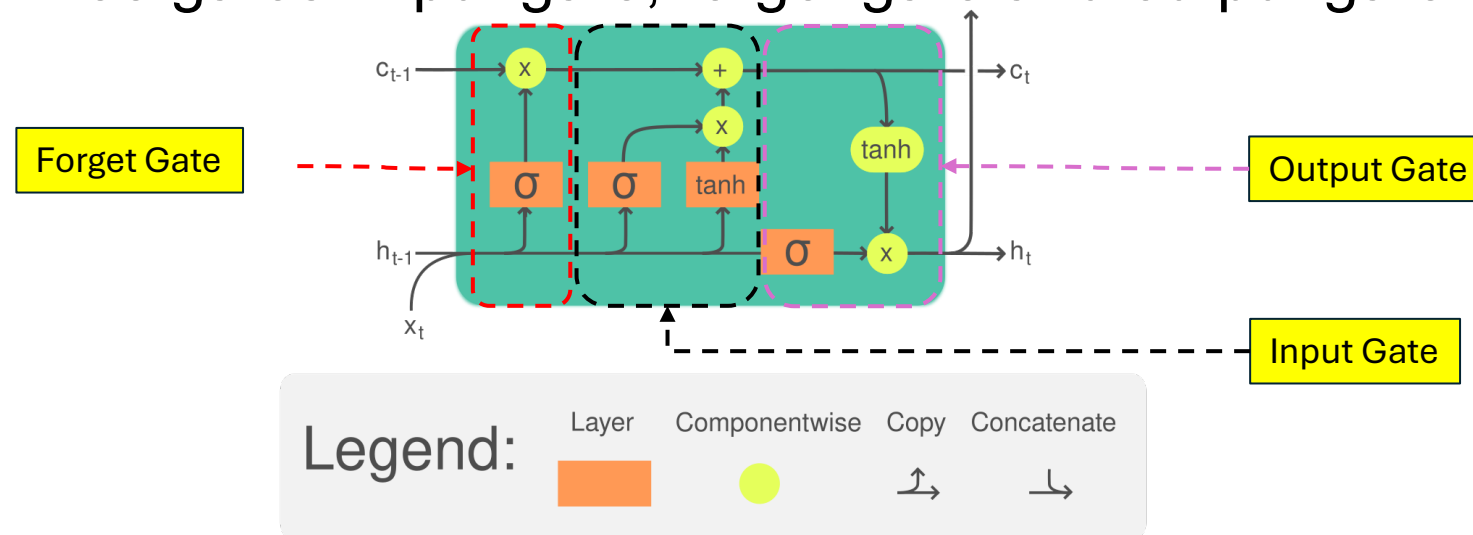
- **Recurrent Neural Networks (RNNs)**
 - Designed to process sequential data one element at a time while maintaining an internal state that summarizes the history of previous inputs.
 - Have capacity to capture temporal dependencies through feedback loop.
 - Memory-like capability helps to retain context and information from earlier elements in the sequence, influencing the generation of subsequent outputs.
 - Challenges faced:
 - Vanishing gradient – gradients used to update the network weights become so small during training, making it difficult to learn long-term dependencies effectively.
 - Inherently sequential – computationally expensive and challenging to parallelize.



Source: https://en.wikipedia.org/wiki/Recurrent_neural_network

LSTM (Before Transformer ...)

- Long short-term memory networks are a specialized type of RNN.
- Designed to address the vanishing gradient problem and capture long-term dependencies in sequential data.
- Ability to incorporate a memory cell that can selectively retain or forget information over time.
- Control by three gates: input gate, forget gate and output gate.

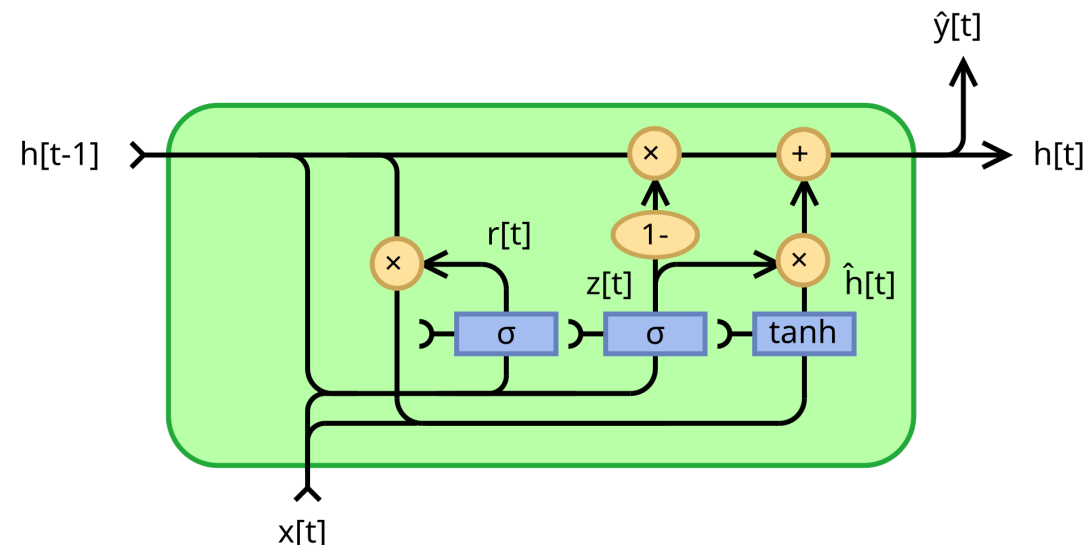


Source: https://en.wikipedia.org/wiki/Long_short-term_memory

Paper: <https://www.bioinf.jku.at/publications/older/2604.pdf>

GRU (Before Transformer ...)

- **Gated Recurrent Unit** networks are a type of neural network architecture commonly used in NLP.
- Also designed to address the vanishing gradient problem.
- Main advantage of GRUs over LSTMs lies in its simpler design and fewer parameters.
- Simpler design makes its faster to train and more straightforward to deploy.



Source: https://en.wikipedia.org/wiki/Gated_recurrent_unit

RNN, LSTM, GRU and Transformer

Description	Recurrent Neural Network (RNN)	Long Short Term Memory (LSTM)	Gated Recurrent Unit (GRU)	Transformers
Overview	RNNs are foundational sequence models that process sequences iteratively, using the output from the previous step as an input to the current step.	LSTMs are an enhancement over standard RNNs, designed to better capture long-term dependencies in sequences.	GRUs are a variation of LSTMs with a simplified gating mechanism.	Transformers move away from recurrence and focus on self-attention mechanisms to process data in parallel
Key characteristics	<ul style="list-style-type: none"> - Recurrent connections allow for the retention of "memory" from previous time steps. 	<ul style="list-style-type: none"> - Uses gates (input, forget, and output) to regulate the flow of information. - Has a cell state in addition to the hidden state to carry information across long sequences. <p>© AIMA.com Research</p>	<ul style="list-style-type: none"> - Contains two gates: reset gate and update gate. - Merges the cell state and hidden state. 	<ul style="list-style-type: none"> - Uses Self-attention mechanisms to weigh the importance of different parts of the input data. - Consists of multiple encoder and decoder blocks. - Processes data in parallel rather than sequentially.
Advantages	<ul style="list-style-type: none"> - Simple structure. - Suitable for tasks with short sequences. 	<ul style="list-style-type: none"> - Can capture and remember long-term dependencies in data. - Mitigates the vanishing gradient problem of RNNs. 	<ul style="list-style-type: none"> - Fewer parameters than LSTM, often leading to faster training times. - Simplified structure while retaining the ability to capture long-term dependencies. 	<ul style="list-style-type: none"> - Can capture long-range dependencies without relying on recurrence - Highly parallelizable, leading to faster training on suitable hardware.
Disadvantages	<ul style="list-style-type: none"> - Suffers from the vanishing and exploding gradient problem, making it hard to learn long-term dependencies - Limited memory span 	<ul style="list-style-type: none"> - More computationally intensive than RNNs - Complexity can lead to longer training times. 	<ul style="list-style-type: none"> - Might not capture long-term dependencies as effectively as LSTM in some tasks. 	<ul style="list-style-type: none"> - Requires a large amount of data and computing power for training. - Can be memory-intensive due to the attention mechanism, especially for long sequences.
Use Cases	<p>Due to its limitations, plain RNNs are less common in modern applications.</p> <p>Used in simple language modeling, time series prediction</p>	Machine translation, speech recognition, sentiment analysis, and other tasks that require understanding of longer context.	Text generation, sentiment analysis, and other sequence tasks where model efficiency is a priority.	<ul style="list-style-type: none"> - State-of-the-art performance in various NLP tasks, including machine translation, text summarization. - Forms the backbone for models like BERT and GPT.
Model variants	Vanilla RNN, Bidirectional RNN, Deep (Stacked) RNN	Vanilla LSTM, Bidirectional LSTM, Peephole LSTM, Deep (Stacked) LSTM	GRU	Original Transformer (Seq-to-Seq), Encoder only (Eg: BERT), Decoder only (Eg: GPT), Text to Text (Eg: T5)

Source: <https://aiml.com/compare-the-different-sequence-models-rnn-lstm-gru-and-transformers/>

The Transformer Architecture

- Transformers are a revolutionary architecture which was introduced in 2017 by Vaswani et al. 2017 in the paper “Attention is All You Need”.
- Transformers are superior to LSTMs (Long-Short-Term-Memory networks) and RNNs (Recurrent Neural Networks) for several reasons.
- Transformers outperform RNNs and LSTMs and excel in parallelism, long-range dependency handling and scalability.
- This makes the architecture of choice for many modern NLP tasks.

[Paper: \[1706.03762\] Attention Is All You Need](#)

Why Transformer?

Advantages:

- Process data in parallel rather than sequentially.
- Able to capture long-range dependencies without relying on recurrence.
- Highly parallelizable leading to faster training on suitable hardware.
- SOTA performance for tasks like translation, text summarization etc.

Disadvantages:

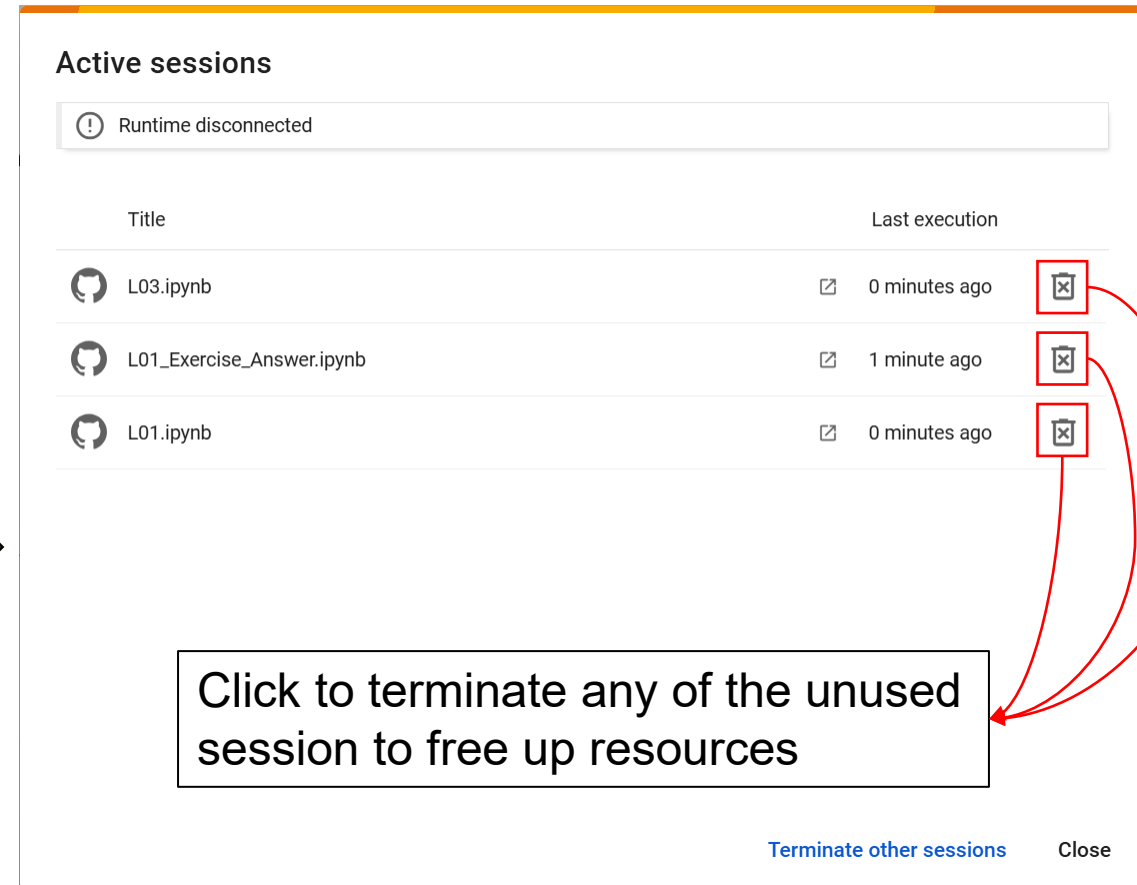
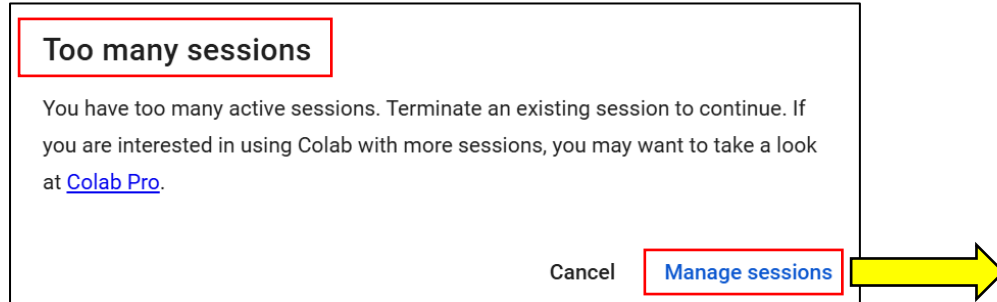
- Memory intensive due to the attention mechanism especially for long sequences.
- Computationally expensive and require a large amount of data for training effectively.

Development Environment

- Basically, you have at least 2 choices to run the code. The most obvious choices are:
 - Google Colab
 - Local machine
- You are encouraged to run the code in Google Colab if you are inexperienced in setting up the development environment.
 - Colab is free and accessible.
 - Less hardware constraint.
 - Seamless integration to Google drive allows you to store and access file easily.
- However, if you want to learn more about the broader ecosystem and intricacies of Python development, setting it up on your local machine gives you full control and comprehensive learning.

Google Colab

- If you choose to run the code in Google Colab (wise choice!), sometimes you may encounter error message like:



Thank you!