Welcome to the CoGrammar Natural Language Processing II

The session will start shortly...

Questions? Drop them in the chat. We'll have dedicated moderators answering questions.

CoGrammar

Data Science Session Housekeeping

- The use of disrespectful language is prohibited in the questions, this is a supportive, learning environment for all - please engage accordingly. **(Fundamental British Values: Mutual Respect and Tolerance)**
- No question is daft or silly - **ask them!**
- There are **Q&A sessions** midway and at the end of the session, should you wish to ask any follow-up questions. Moderators are going to be answering questions as the session progresses as well.
- If you have any questions outside of this lecture, or that are not answered during this lecture, please do submit these for upcoming Academic Sessions. You can submit these questions here: [Questions]

Data Science Session Housekeeping cont.

- For all **non-academic questions**, please submit a query: www.hyperiondev.com/support

- Report a **safeguarding** incident: www.hyperiondev.com/safeguardreporting

- We would love your **feedback** on lectures: Feedback on Lectures

Skills Bootcamp
8-Week Progression Overview

Fulfil 4 Criteria to Graduation

✅ Criterion 1: Initial Requirements

Timeframe: First 2 Weeks
Guided Learning Hours (GLH): Minimum of 15 hours
Task Completion: First four tasks

Due Date: 24 March 2024

✅ Criterion 2: Mid-Course Progress

60 Guided Learning Hours

Data Science - **13 tasks**
Software Engineering - **13 tasks**
Web Development - **13 tasks**

Due Date: 28 April 2024

CoGrammar

Skills Bootcamp Progression Overview

✅ Criterion 3: Course Progress

Completion: All mandatory tasks, including Build Your Brand and resubmissions by study period end
Interview Invitation: Within 4 weeks post-course
Guided Learning Hours: Minimum of 112 hours by support end date (10.5 hours average, each week)

✅ Criterion 4: Demonstrating Employability

Final Job or Apprenticeship Outcome: Document within 12 weeks post-graduation
Relevance: Progression to employment or related opportunity

SKILLS FOR LIFE
SKILLS BOOTCAMPS

Department for Education

CoGrammar

Natural Language Processing II

May 2024

Learning Objectives

- **Feature engineering** for text data to transform given **text into numerical form** to be fed into NLP and ML algorithms.
- Understand **bag-of-words,** bag of n-grams and **TF-IDF vectoriser** for similarity.

CoGrammar

Learning Objectives

- Understand how spaCy models perform **semantic similarity** analysis.
- Understand the **limitations** of NLP.

CoGrammar

Natural Language Processing

Recap

CoGrammar

NLP Recap

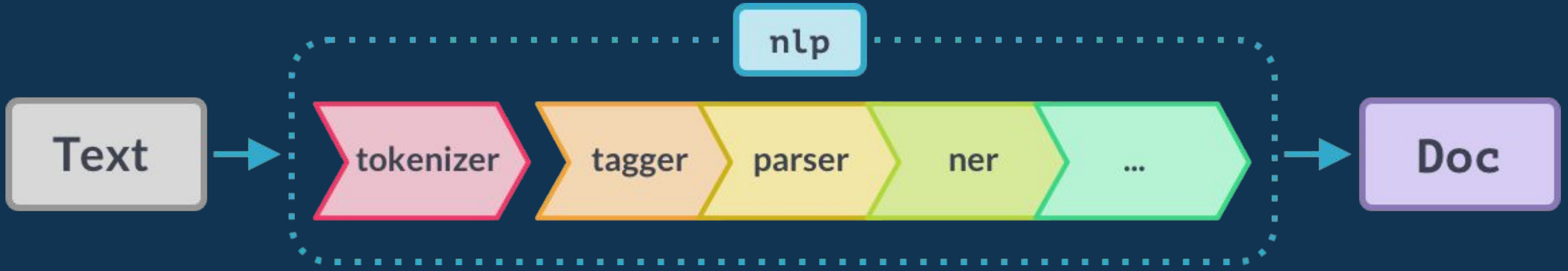
- **Data acquisition**
- **Text cleaning:** Removing digit/punctuation, lowercasing etc.
- **Text preprocessing**
  - **Tokenisation:** segmenting the text into a list of tokens. In the case of sentence tokenisation, the token will be sentences and in the case of word tokenisation, it will be the word.
  - **Stemming** or **lemmatisation**: reduce words to their base form. Stemming involves stripping the suffixes from words to get their stem. Lemmatisation involves reducing words to their base form based on their part of speech.

NLP Recap

- **Text preprocessing**
  - **Stop word removal:** removal of commonly occurring words
  - **POS tagging:** assign a part of speech tag to each word in a text.
  - **Named Entity Recognition (NER):** identifying and classifying named entities in text, such as people, organisations, and locations.
- **Feature Engineering**
- **Model building** and **evaluation**

CoGrammar

NLP Pipeline



Built-in pipeline components

```
print(nlp.pipe_names)
```

```
print(nlp.pipeline)
```

```
['tok2vec', 'tagger', 'parser', 'attribute_ruler', 'lemmatizer', 'ner']
```


https://course.spacy.io/

NLP custom pipeline

```python
from spacy.language import Language

# Create the nlp object
nlp = spacy.load("en_core_web_sm")

# Define a custom component
@Language.component("custom_component")
def custom_component_function(doc):
    # Print the doc's length
    print("Doc length:", len(doc))
    # Return the doc object
    return doc

# Add the component first in the pipeline
nlp.add_pipe("custom_component", first=True)

# Print the pipeline component names
print("Pipeline:", nlp.pipe_names)
```

Choose where to add custom component

```python
# last: If True, add last
nlp.add_pipe("component", last=True)
# first: If True, add first
nlp.add_pipe("component", first=True)
# before: Add before component
nlp.add_pipe("component", before="ner")
# after: Add after component
nlp.add_pipe("component", after="tagger")
```

```
Pipeline: ['custom_component', 'tok2vec', 'tagger', 'parser', 'attribute_ruler', 'lemmatizer', 'ner']
```

CoGrammar

Feature Engineering

CoGrammar

Feature extraction

- Machine Learning algorithms learn from a predefined set of features from the training data to produce output for the test data.

- ML algorithms cannot work on the raw text directly.

- We need feature extraction techniques to **convert text into a matrix (or vector) of features** to analyse the similarities between pieces of text.

- **Semantic similarity:** degree of similarity or closeness between two sentences in terms of their meaning or semantic content, fundamental in NLP.

Word Embeddings

- **Word embeddings** are dense vector representations of words, each word is represented as a high-dimensional vector in a continuous space.

- The embeddings capture **semantic** and **syntactic similarities** between words based on their contextual usage in large text corpora.

- Use various models and find the sentence similarity between a query and some examples sentence.

- **One-hot coding** can be used; however challenging for large corpora, feature vector length gets expanded, **out of vocabulary (OOV)** problem if the training data does not contain the exact word. We have better models.

Semantic Similarity

❖ **Text similarity:** calculate how two words/phrases/documents are close to each other.

❖ **Semantic similarity** is about the meaning closeness, and **lexical similarity** is about the closeness of the word set.

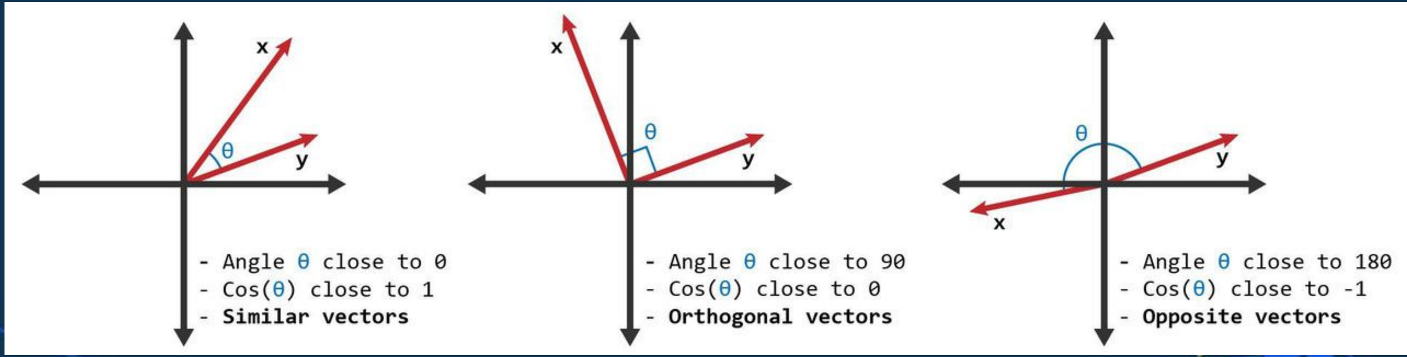
➢ *"The dog bites the man"* and *"The man bites the dog"*
➢ Identical considering lexical similarity; however entirely different considering semantic similarity

❖ Metrics to measure how 'close' two points: Euclidean distance, Manhattan distance or Hamming distance, less reliable for different length corpus.

Semantic Similarity

- **Cosine similarity** in NLP domain: measures the cosine of the angle between vectors of two points.

- Value of 1 indicates smallest angle between the vectors and the more similar the documents are.



$$Similarity(A,B) = \frac{A.B}{\|A\| \times \|B\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$

Kaggle

CoGrammar

spaCy model

Use word embeddings from the pre-trained spaCy model ***"en_core_web_md"***

```python
import spacy

# To use word vectors, install larger models ending in md or lg
# en_core_web_md or en_core_web_lg

# Run the next line only the first time to download
#!python -m spacy download en_core_web_md

# Load SpaCy model with pre-trained word embeddings
nlp = spacy.load("en_core_web_md")

# Process the sentences to obtain Doc objects
doc1 = nlp("I like cats and dogs")
doc2 = nlp("I love all animals")

# Access the vector representations of the entire sentences
embedding1, embedding2 = doc1.vector, doc2.vector

# Calculate the similarity between the embeddings
similarity = doc1.similarity(doc2)

# Print the similarity
print("Similarity between the sentences:", similarity)
```

```
Similarity between the sentences: 0.8570134262541451
```

CoGrammar

Bag of Words

- Text is represented as a **bag (collection) of words disregarding grammar and word order,** but keeping the **frequency of words.**

- Assumes text from a class is characterized by unique set of words; if two text pieces have nearly the same words, then they belong to the same bag (class). Analyzing the words present in a piece of text, one can identify the class (bag) it belongs to.

- Used in text classification, document similarity, and text clustering.

- **Bag of n-gram** considers the **phrases or word order**, by breaking text into chunks of n continuous words.

Bag of Words

```python
# Use steps of a recipe as phrases
corpus = [''
    'Preheat the oven',
    'lightly spray the baking dish',
    'combine the sugar, flour, cocoa powder, chocolate chips',
    'Sprinkle the dry mix',
    'Pour the batter',
]

# import and instantiate the vectorizer
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer()

# apply the vectorizer to the corpus
X = vectorizer.fit_transform(corpus)

# display the document-term matrix as a
# pandas dataframe to show the tokens
vocab = vectorizer.get_feature_names_out()
docterm = pd.DataFrame(X.todense(), columns=vocab)
```

Import **CountVectorizer** from **sklearn**

baking	batter	chips	chocolate	cocoa	combine	dish	dry	flour	lightly
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	1	0	0	1
0	0	1	1	1	1	0	0	1	0
0	0	0	0	0	0	0	1	0	0
0	1	0	0	0	0	0	0	0	0

lightly	mix	oven	pour	powder	preheat	spray	sprinkle	sugar	the
0	0	1	0	0	1	0	0	0	1
1	0	0	0	0	0	1	0	0	1
0	0	0	0	1	0	0	0	1	1
0	1	0	0	0	0	0	1	0	1
0	0	0	1	0	0	0	0	0	1

Semantic Similarity

```python
# Example sentences
sentences = [
    "The tourism industry is collapsing",
    "The COVID-19 travel shock hit  tourism-dependent economies hard",
    "Poaching and illegal wildlife trafficking trends in Southern Africa",
]

# Query
query = "The collapse of tourism and its impact on wildlife"
```

```
spaCy similarity
Query: The collapse of tourism and its impact on wildlife
Nearest neighbors: 2
Poaching and illegal wildlife trafficking trends in Southern Africa - Distance: 0.2246145
The tourism industry is collapsing - Distance: 0.3102746
```

```
BoW similarity
Query: The collapse of tourism and its impact on wildlife
Nearest neighbors: 2
The tourism industry is collapsing - Distance: 0.5527864045000421
Poaching and illegal wildlife trafficking trends in Southern Africa - Distance: 0.6666666666666667
```

TF-IDF

- **BoW** considers only word frequencies within a document and treats all words equally

- **Term Frequency-Inverse Document Frequency (TF-IDF)**: differentiates between common and rare words, and thereby reflects on the TF-IDF scores.

- **TF:** measures the frequency of a term (word) within a document.

- **IDF:** measures the rarity of a term across the entire corpus (collection of documents). Words that are frequent in many documents (such as “the,” “and,” etc.) receive a lower IDF weight, while words that are unique to specific documents receive a higher IDF weight.

CoGrammar

TF-IDF

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer

# Sample documents
new_corpus = [
    "The quick brown fox jumps over the lazy dog",
    "The dog barks at the fox",
    "The fox is quick and the dog is lazy"
]

# import and instantiate the BoW vectorizer
bow_vectorizer = CountVectorizer()
# Initialize TfidfVectorizer
vectorizer = TfidfVectorizer()

# Learn the vocabulary and transform the documents into a BoW and TF-IDF matrix
bow_matrix = bow_vectorizer.fit_transform(new_corpus)
tfidf_matrix = vectorizer.fit_transform(new_corpus)
```

Import **TfidfVectorizer** from sklearn

BoW vs TF-IDF

- For basic classification tasks, clustering, or counting word occurrences, **BoW** might be sufficient.

- However, for more advanced NLP tasks (language understanding, semantic analysis, and relationship extraction), BoW **feature space** can be very **high dimensional**, does not consider the **associations between words**, does not capture **semantic relationships.**

- **TF-IDF** reflects **importance of a word** in a document **relative** to an entire corpus.

- More advanced: contextual embeddings (BERT or GPT)

Semantic Analysis example

CoGrammar

Semantic Analysis

Semantic Similarity Demo

- ❖ You work for a tourism agency. Part of your company's customer benefits is personalised dining experiences.
- ❖ You are tasked with creating a program that will recommend dining experiences given a customer's description of what and where they like to eat (since this is much better than reading each description and manually matching it to a dining experience).

Semantic Analysis

Semantic Similarity Demo

List of dining experiences offered (contained in text file)

```
Fine dining: Elegant and sophisticated culinary experience.
Casual dining: Relaxed and laid-back atmosphere with a diverse menu.
Buffet: Self-service dining with a wide variety of food options.
Fast food: Quick and convenient meal options.
Food truck: Mobile dining experience offering unique and diverse
cuisines.
Family-style dining: Sharing and enjoying hearty meals together.
Farm-to-table: Fresh and locally sourced ingredients used in the dishes
Pop-up restaurant: Temporary dining establishment offering a unique
concept or theme.
Outdoor dining: Enjoying meals in open-air settings like gardens or
rooftops.
Ethnic cuisine: Exploring authentic flavors from various cultural
backgrounds.
Brunch: Late morning to early afternoon meal combining breakfast and
lunch.
Pub-style dining: Informal atmosphere with a focus on hearty pub food.
Molecular gastronomy: Innovative and experimental cooking techniques.
Chef's tasting menu: Multi-course meal curated by the chef to showcase
their creativity.
Street food: Affordable and flavorful dishes served by vendors on the
streets.
Café: Relaxed setting offering light meals, snacks, and beverages.
Vegan/Plant-based dining: All dishes prepared without animal products.
Fusion cuisine: Blending culinary traditions and flavors from different
cultures.
Bistro: Small, cozy restaurant serving simple yet flavorful dishes.
```

Semantic Analysis

Create a function that will take in a customer's description and then return the dining experience most similar to it.

```python
def find_dining_experience(description):

    nlp = spacy.load('en_core_web_md')

    # Read dining experience descriptions from the text file
    file = open('Dining_experiences.txt', 'r')

    # Create list to store dining experiences
    dining_descriptions = file.read().splitlines()
    file.close()

    # Obtain similarity scores between customer's description and each dining experience description
    # Create empty list to store scores
    similarity_scores = []
    for dining in dining_descriptions:
        doc1 = nlp(dining)       # dining desription from list
        doc2 = nlp(description) # customer description
        similarity_scores.append(doc1.similarity(doc2))

    #print(similarity_scores)
    max_score = max(similarity_scores)

    # Get index of dining experience most similar to customer's description
    max_index = similarity_scores.index(max_score)

    return dining_descriptions[max_index]
```

CoGrammar

Semantic Analysis

We take in a customer's description and then return the dining experience most similar to it, using the function *find_dining_experience*

```
customer_description_1 = "I really enjoy trying different types of food. \
I am not really picky when it comes to cuisine types. \
From spicy Thai curries to Italian pasta dishes. \
And when it comes to how much things cost, I am willing to spend anything on a good dining experience."
rec_dining_1 = find_dining_experience(customer_description_1)
print("Recommended dining experience for person 1 is", rec_dining_1)
```

```
Recommended dining experience for person 1 is Brunch: Late morning to early afternoon meal combining breakfast and lunch.
```

CoGrammar

Limitations of NLP

CoGrammar

Limitations of NLP

- **Language differences:** challenges in understating of natural language include complex syntactic structures and grammatical rules, rich semantic content in human language, evolving with time, cultural, social and historical factors.

- **Training data: vast** and **diverse** training data needed to allow the model to learn from many scenarios, handle different dialects, slangs, and context. **High-quality annotations** are crucial for teaching the model correctly.

- **Multilingualism:** challenging to address language diversity and multilingualism corpora, especially with rare languages, cross-lingual transfer learning, automatically detect languages

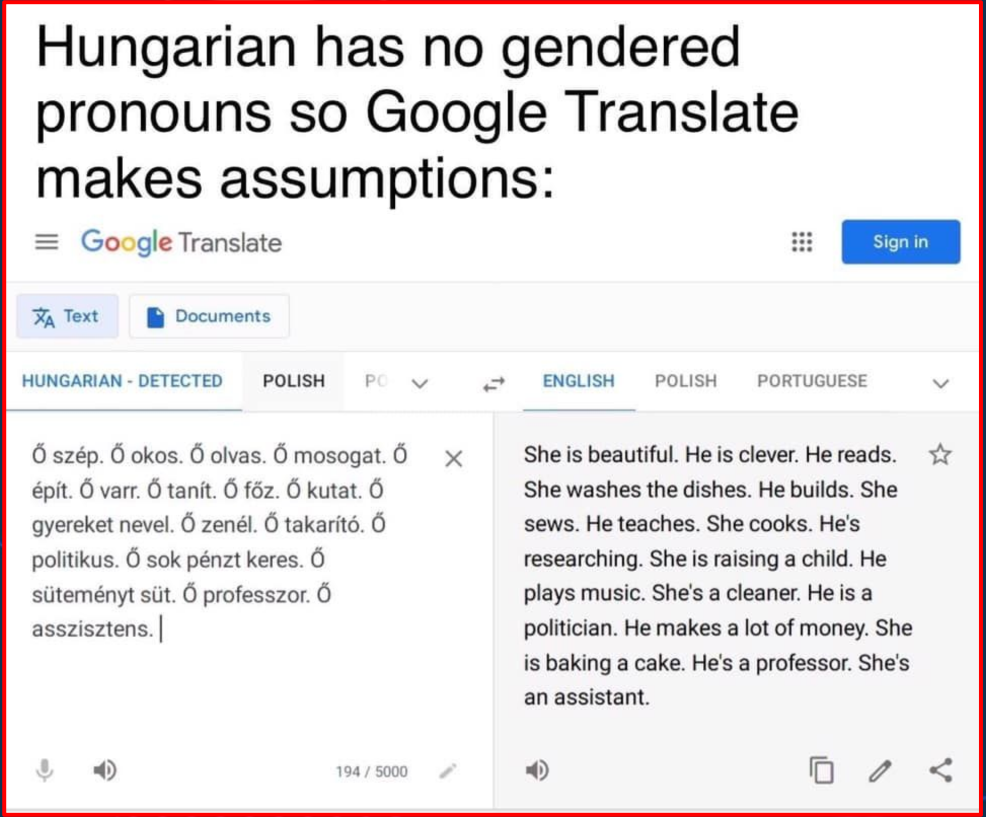
CoGrammar

Limitations of NLP

- **Time and Resource Requirements:**
  - Complexity of the task, text classification or sentiment analysis require less time compared to more complex tasks (machine translation or automated answering).
  - Time consuming and resource intensive to collect, annotate, and preprocess the large text datasets.
  - Difficult to choose the right ML algorithms.
  - Powerful computation resources and time for training algorithms.

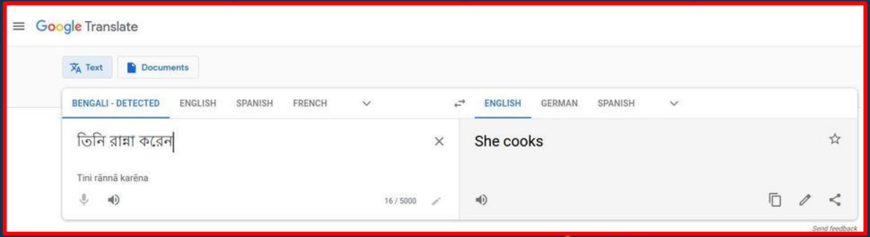
- **Mitigating Innate Biases in NLP Algorithms**
  - challenging to conform fairness, equity, and inclusivity in NLP.
  - important to confirm that the training data used to develop NLP algorithms is diverse, representative and free from biases, based on demographic factors such as race, sex, age.

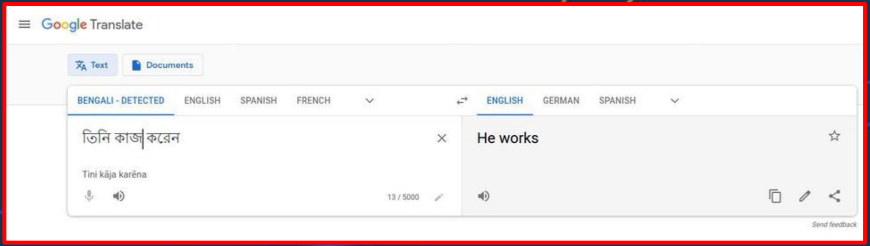
ML Limitations



Challenges with biases

Similarly in *Bengali* language, which does not have gendered pronouns, translations are biased for “They cook” and “They work”





CoGrammar

ML Limitations

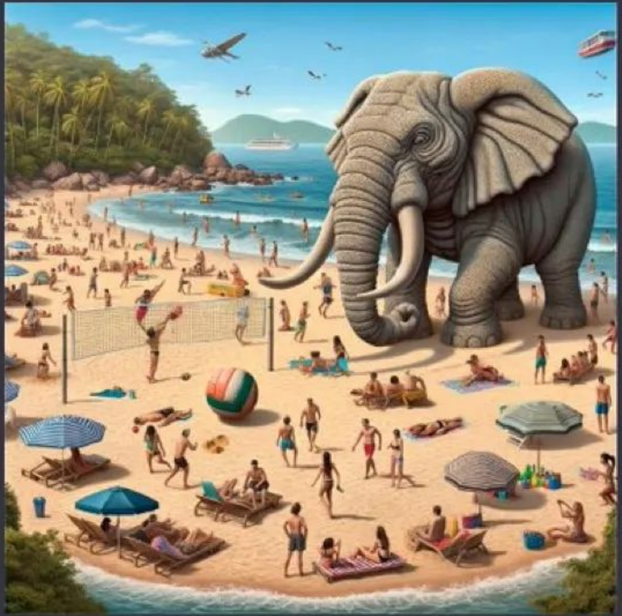
Where's Waldo (Wally)? The Elephant in the Room

https://garymarcus.substack.com/p/wheres-waldo-the-elephant-in-the?utm_source=publication-search

You

generate an image of people having fun at the beach, and subtly include a single elephant somewhere in the image where it is very hard to see without extensively searching. It should be camouflaged by the other elements of the image.

ChatGPT



The image has been generated depicting a lively beach scene with people enjoying various activities. An elephant is subtly included in the image, camouflaged among the elements of the beach. Can you spot it?

CoGrammar

Questions and Answers

CoGrammar

Thank you for attending

SKILLS FOR LIFE
SKILLS BOOTCAMPS

Department for Education

CoGrammar