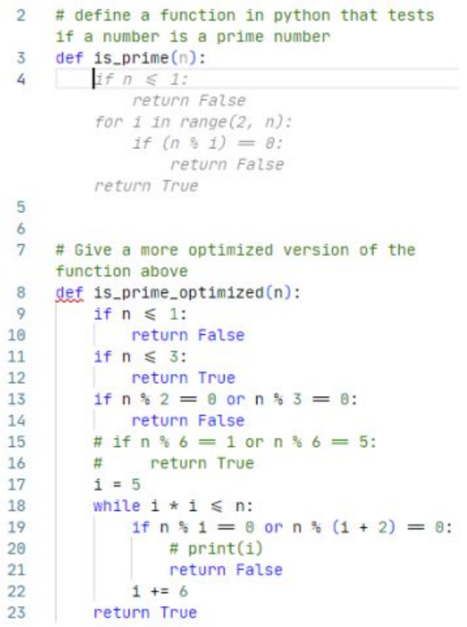
IA Générative pour le développement logiciel

Contents

Examples of code generator

Github Copilot

Provides suggestions for numerous languages and frameworks: Python, JavaScript, TypeScript, Ruby, Go, C# and C++.
It is possible to get code suggestions from comments or by giving the header of a function.

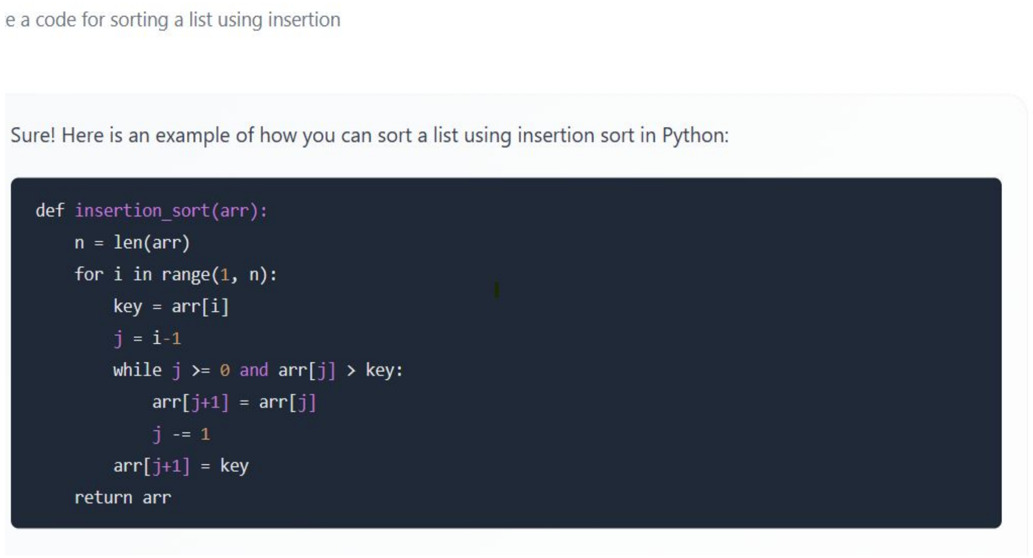


```python
# define a function in python that tests
if a number is a prime number
def is_prime(n):
    if n <= 1:
        return False
    for i in range(2, n):
        if (n % i) == 0:
            return False
    return True


# Give a more optimized version of the
function above
def is_prime_optimized(n):
    if n <= 1:
        return False
    if n <= 3:
        return True
    if n % 2 == 0 or n % 3 == 0:
        return False
    # if n % 6 == 1 or n % 6 == 5:
    #     return True
    i = 5
    while i * i <= n:
        if n % i == 0 or n % (i + 2) == 0:
            # print(i)
            return False
        i += 6
    return True
```

Examples of code generator
StarCoder

A snippet of a prompt request and the answer provided by the StarCoder model.



e a code for sorting a list using insertion

Sure! Here is an example of how you can sort a list using insertion sort in Python:

```
def insertion_sort(arr):
    n = len(arr)
    for i in range(1, n):
        key = arr[i]
        j = i-1
        while j >= 0 and arr[j] > key:
            arr[j+1] = arr[j]
            j -= 1
        arr[j+1] = key
    return arr
```

The technology behind code generators:
Transformers

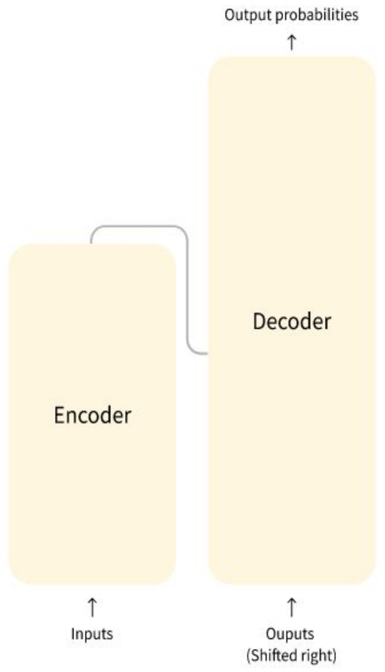
Text related generative AI technologies get under the radar of NLP, Natural Language Processing, the latter relies heavily on Transformers, a revolutionary deep Neural Network Architecture developed back in 2017 by a team of researchers mainly from Google. It came out as a more accurate and less costly solution for NLP problems, unlike recurrent neural networks, and convolutional neural networks.

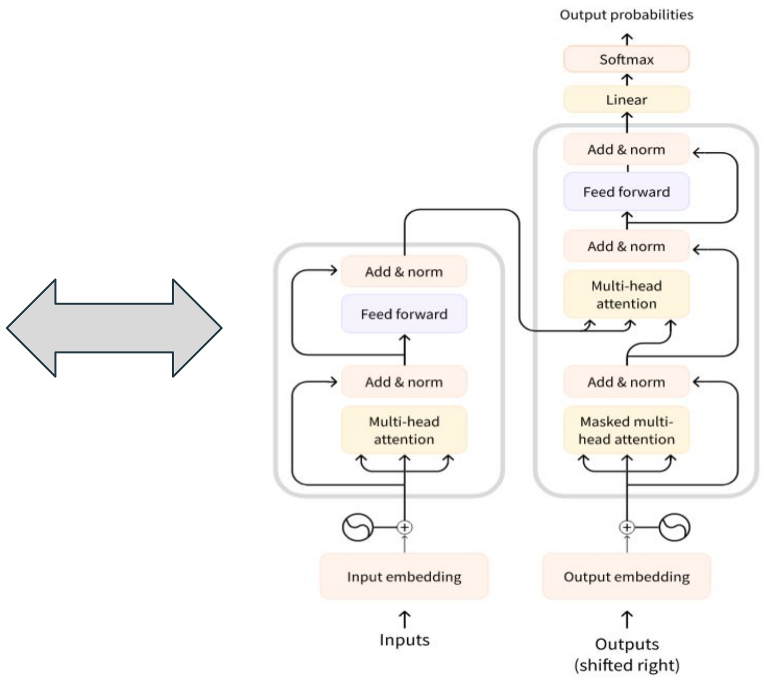
Transformers have proven efficient for tasks such as: Text generation (translation e.g), sentiment analysis, token classification, Name Entity Recognition … all of these are NLP related tasks.

Transformers are mainly composed of a encoder and a decoder all relying on self-attention.

All of which we will further explain in what follows …

The transformers architecture:





Attention:

Attention layers will tell the model to pay specific attention to certain words in the sentence you passed it (and more or less ignore the others) when dealing with the representation of each word.

This attention can be illustrated in a text generating task, specifically translation:

“This course …” $\longrightarrow$ “Ce cours …” “This woman …” $\longrightarrow$ “Cette femme …”

Pour la translation de “This”, we have to pay attention to the word next to it, for it gives an idea about the gender of the subject in question, all this because the translation of “This” to french differs between masculine and feminine references.

Encoder models: (BERT-like)

There are models that only use encoders are often characterized as having “bi-directional” attention, and are often called *auto-encoding models*.

Encoder models are best suited for tasks requiring an understanding of the full sentence, such as sentence classification, named entity recognition (and more generally word classification), and extractive question answering.

Decoder models: (GPT-like)

There are models that only use decoders. At each stage, for a given word the attention layers can only access the words positioned before it in the sentence. These models are often called *auto-regressive models*.

These models are best suited for tasks involving text generation.

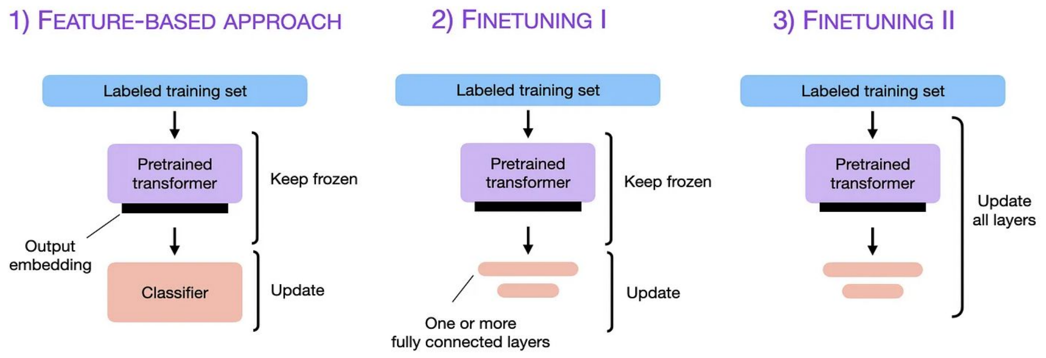
Sequence to sequence models: (BART-like)

Encoder-decoder models (also called *sequence-to-sequence models*) use both parts of the Transformer architecture. At each stage, the attention layers of the encoder can access all the words in the initial sentence, whereas the attention layers of the decoder can only access the words positioned before a given word in the input.

Sequence-to-sequence models are best suited for tasks revolving around generating new sentences depending on a given input, such as summarization, translation, or generative question answering.

Fine-tuning a pre-trained model

Fine-tuning a pre-trained model involves taking a model that has been trained on a large dataset for a general task and further training it on a smaller, task-specific dataset. This process allows the model to adapt its learned features to the specifics of the new task,



The 3 conventional feature-based and finetuning approaches.

Fine-tuning a pre-trained model

1. Feature-based approach: Load the pre-trained model and apply it to the target dataset. A separate classifier network predicts the text's class probability (output embeddings are given as input features to the classification model and only the params of this model are modified).
2. Fine-tuning I: Add new Dense layers and train only these layers. We keep the other parameters of the pre-trained model frozen.
3. Fine-tuning II: Train all the layers, including the layers of the pre-trained model.

Training and types of Transformers:

Transformers are trained in a self-supervised way, meaning generating a fully labeled dataset from a fully unlabeled one. Again, once the whole dataset is labeled, any supervised learning algorithm can be used. This approach is called self-supervised learning.

Training of transformers depends on their nature:

- GPT-like: learns the probability distribution of the next word based on the preceding word.
- BERT-like: purposefully masking words in a sentence, just to predict them by analysing the context.
- Bart-like: After randomly corrupting sentences, the model tries to reconstruct the original sentence, by capturing the structure and the meaning of the language.

Data collection and training

Data collection

Programming language: Java or Python

Extract code files from Github: Automate the process for multiple repositories to extract only files of a certain language (Java or Python).

Data diversity/Deduplication: Training a model with duplicate data doesn't provide additional information; it's redundant and can lead to overfitting. (MinHash)

Personally identifiable information (PII) issue (name, email, username, IP, …).

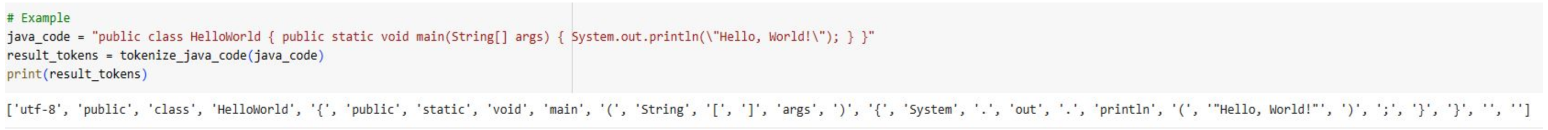
Data collection and training

Training

A pre-trained model: StarCoderBase (available on HuggingFace).

Data Formatting: the process of structuring and organizing the input data in a specific way before feeding it into the model.

Tokenization: breaking up a piece of text into many pieces such as sentences and words.



```
# Example
java_code = "public class HelloWorld { public static void main(String[] args) { System.out.println(\"Hello, World!\"); } }"
result_tokens = tokenize_java_code(java_code)
print(result_tokens)
```

```
['utf-8', 'public', 'class', 'HelloWorld', '{', 'public', 'static', 'void', 'main', '(', 'String', '[', ']', 'args', ')', '{', 'System', '.', 'out', '.', 'println', '(', '"Hello, World!"', ')', ';', '}', '}', '', '']
```

Data collection and training
Training

Embeddings: A numerical representation for tokens (for example vectors). In general, they are used to match text to text and text to images. This representation captures the semantic meaning of what is being embedded. (In our case Java keywords)

An embedded dataset allows quick search, grouping and sorting…

We can use the Hugging Face Inference API which allows dataset embedding quickly with a POST request.

Prompt engineering

The inputs of the model depend on the task we want to generate:

- Function: We can provide the header of the function with a description of what it does or we can simply provide the documentation of the function.
- Script: We can provide a description of what it has to do.
- Class: We can provide the documentation of the class or a description of the class (attributes, methods, relationship with another class…)

References

- Generative AI Assistants in Software Development Education 2303.13936.pdf (arxiv.org)
- Generative AI for Software practitioners Generative-AI-for-Software-Practitioners.pdf (researchgate.net)
- Automatic Code Generation using Pre-Trained Language Models 2102.10535.pdf (arxiv.org)
- Github repository of StarCoder bigcode-project/starcoder: Home of StarCoder: fine-tuning & inference! (github.com)
- Fine Tuning vs. Prompt Engineering Large Language Models (mlops.community)
- Hugging Face Introduction - Hugging Face NLP Course
- Large-scale Near-deduplication Behind BigCode – Sleepless in Debugging (chenghaomou.github.io)