

Large Language Model for Text Analysis

Ahmad Fathan Hidayatullah

fathan@uii.ac.id

Pesantren Sains Data 1445 H

Thursday, 21 March 2024

Outline

- Large Language Models
- Text Analysis
- Case Study

Large Language Models (LLMs)

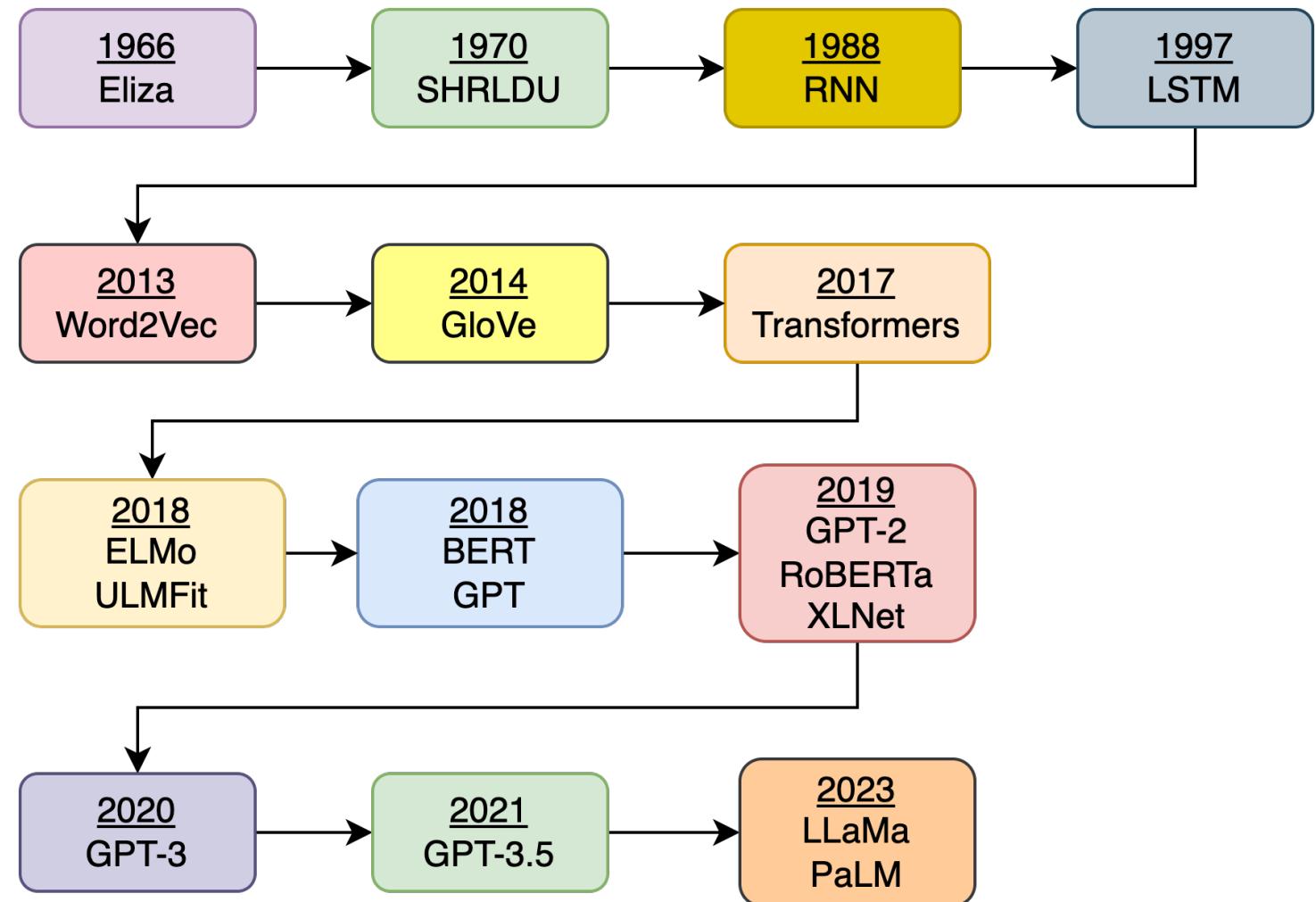


Large Language Models (LLMs)

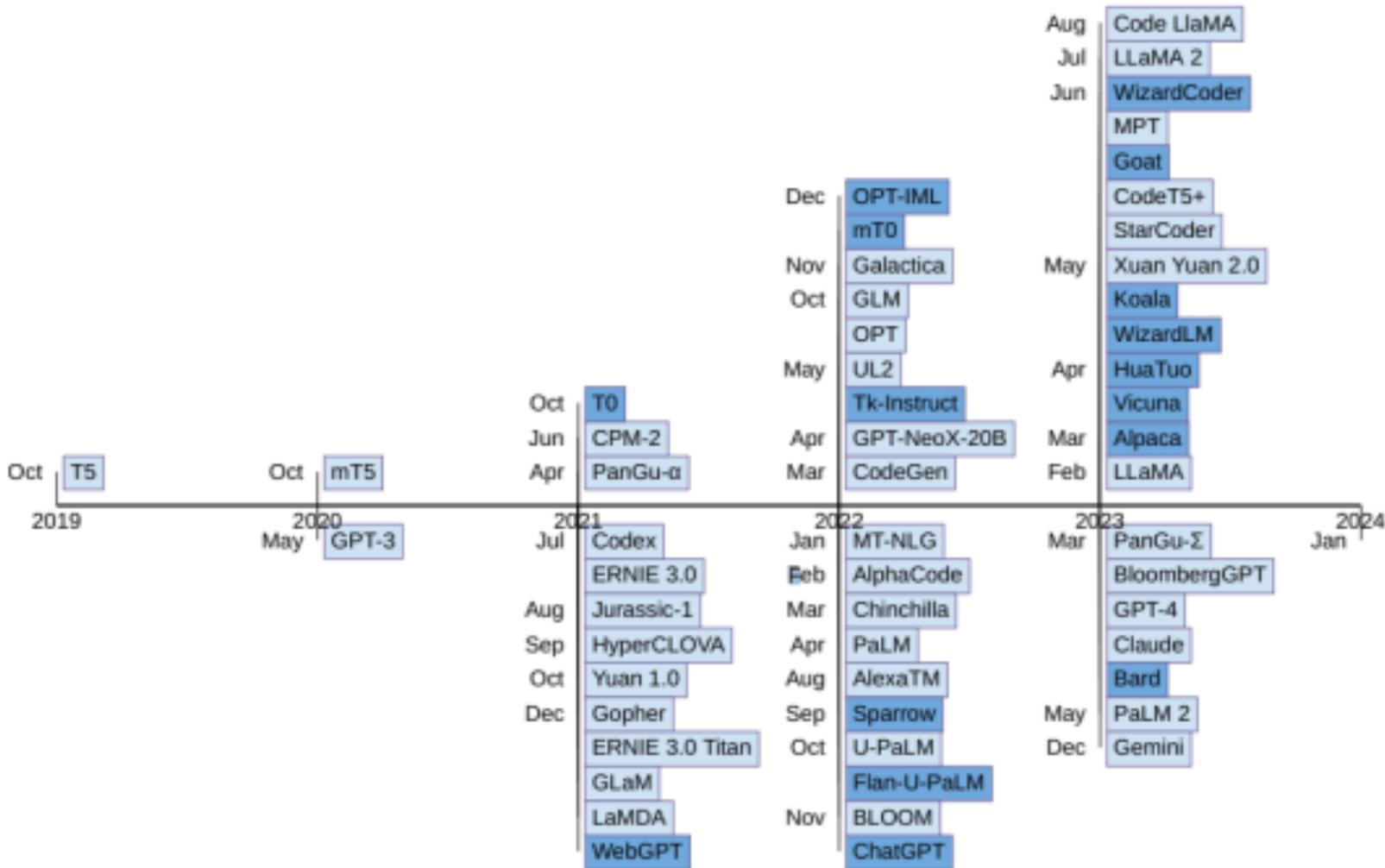
- LLMs are **foundation models** pre-trained on **large** amounts of **unlabeled** and **self-supervised** text data, including books, articles, and conversations.
- LLMs utilize **Transformer-based neural networks** to **learn patterns** and connections within language, encompassing syntactic and semantic structures like grammar, word order, and word meanings.
- The **primary objective** of LLMs is to **replicate** human-like language understanding in machines, focusing specifically on text and text-like inputs.



A Brief History of LLMs



Chronological Display of LLM Releases



Why are they called “Large”?

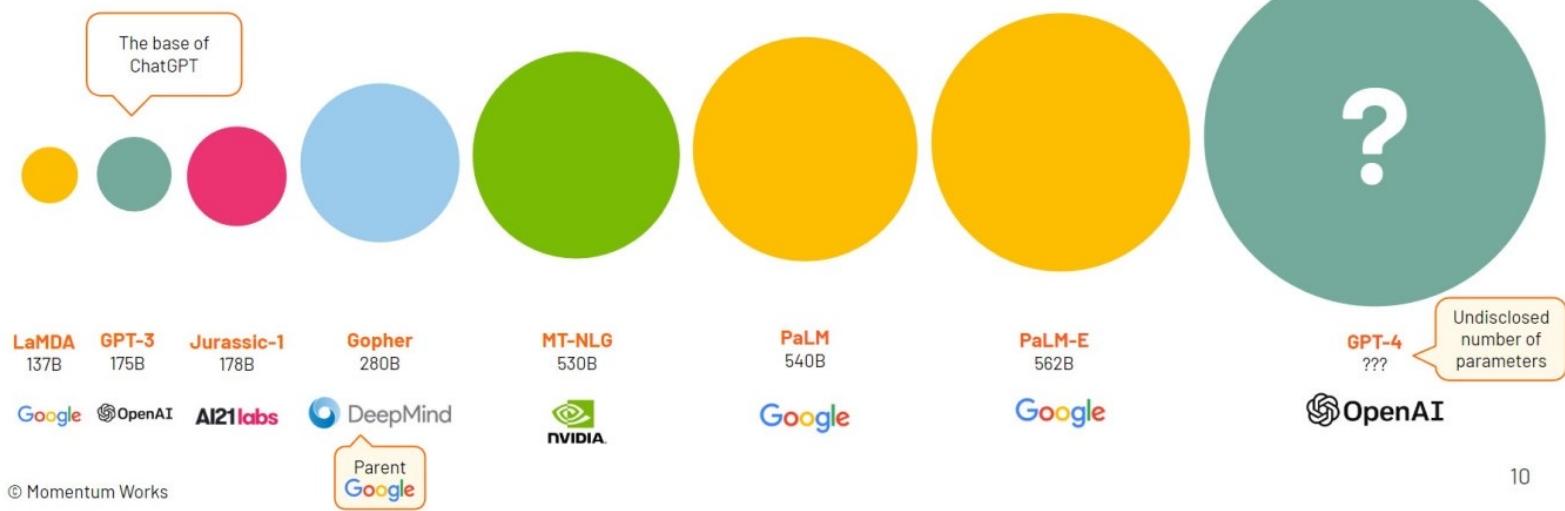
- LLMs are called "large" because they are pre-trained with a large number of **parameters (100M+)** on **large corpora** of text to process/understand and generate natural language text for a wide variety of NLP tasks.

— Large Language Models are becoming very large indeed

Small models (< 100b parameters)



Large models (>100b parameters)

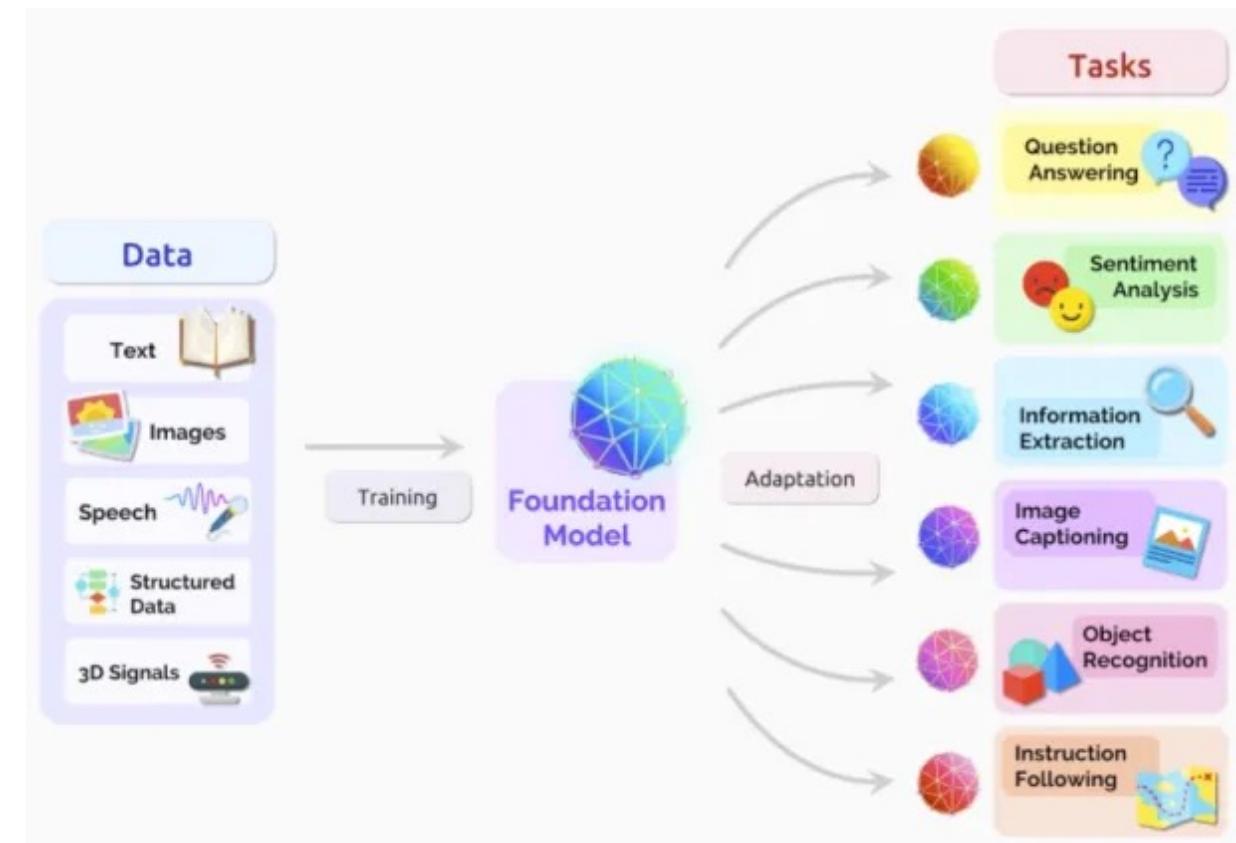


List of Large Language Models Compared to Different Parameters

Model	Disk Size (GB)	Memory Usage (GB)	Parameters (Millions)	Training Data Size (GB)
BERT-Large	1.3	3.3	340	20
GPT-2 117M	0.5	1.5	117	40
GPT-2 1.5B	6	16	1,500	40
GPT-3 175B	700	2,000	175,000	570
T5-11B	45	40	11,000	750
RoBERTa-Large	1.5	3.5	355	160
ELECTRA-Large	1.3	3.3	335	20

LLMs are Task-agnostic Models

- LLMs can solve and perform any tasks, such as answering questions, translating documents, summarizing documents, completing sentences, etc.



Source: <https://research.aimultiple.com/large-language-models/>

LLMs vs Traditional Language Model

LLMs

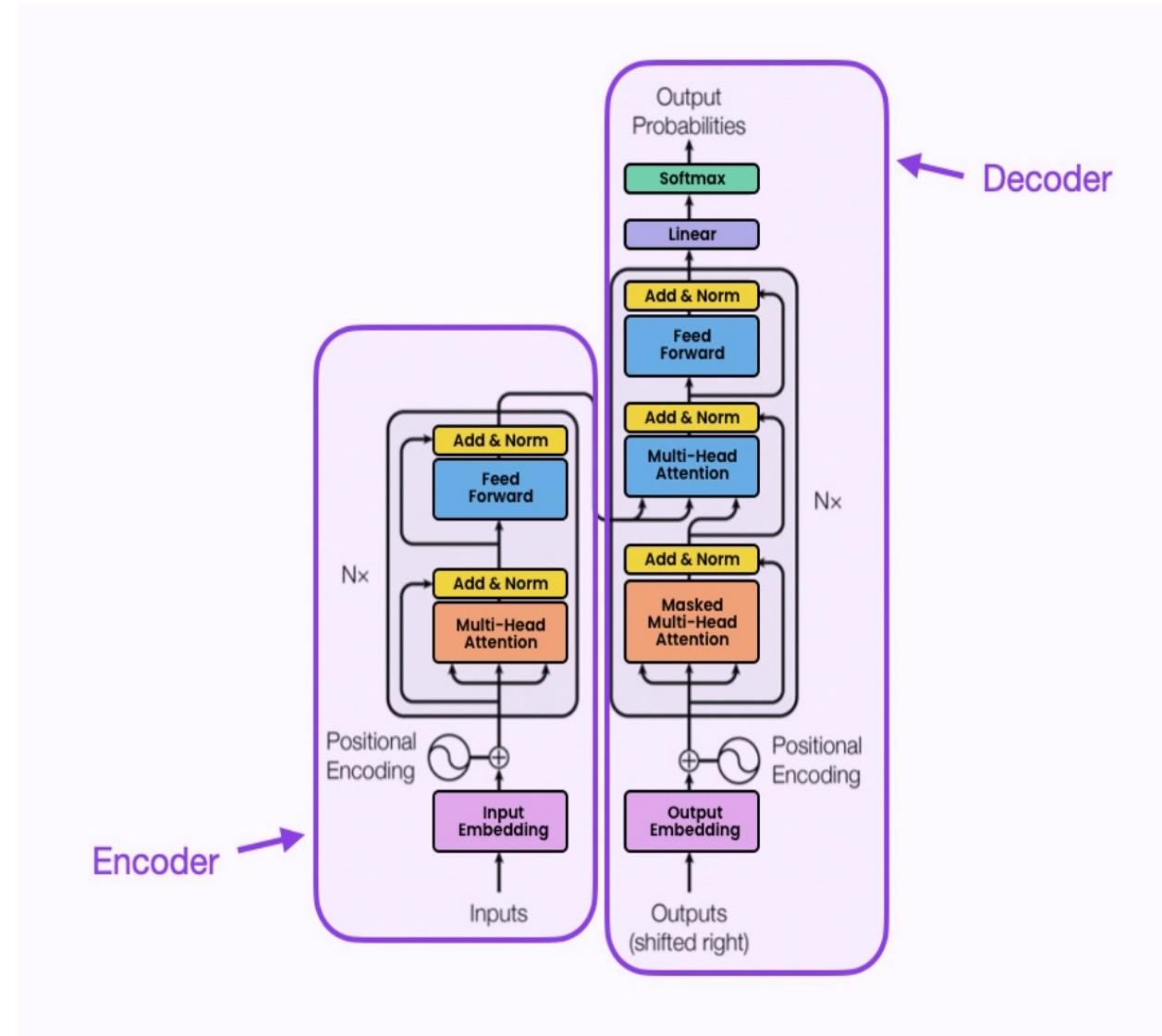
- Based on Transformer architecture and self-attention mechanism
- Learning from extensive text datasets through data-driven approaches
- LLMs consider broader context
- Capable of performing tasks with limited or no task-specific examples
- LLMs can be fine-tuned for a specific task or domain to improve performance
- Adaptable to different languages due to data-driven learning

Traditional Language Model

- Relied on statistical approaches, n-gram models, rules, and hand-crafted linguistic features
- Limited contextual understanding and struggled with complex language structures
- A traditional language model only considers smaller context windows
- It utilizes rule-based syntactic parsers for sentence structure analysis
- Heavily dependent on human experts for linguistic features creation
- Often tailored for specific task, lacking versatility across various contexts

Architecture of LLMs

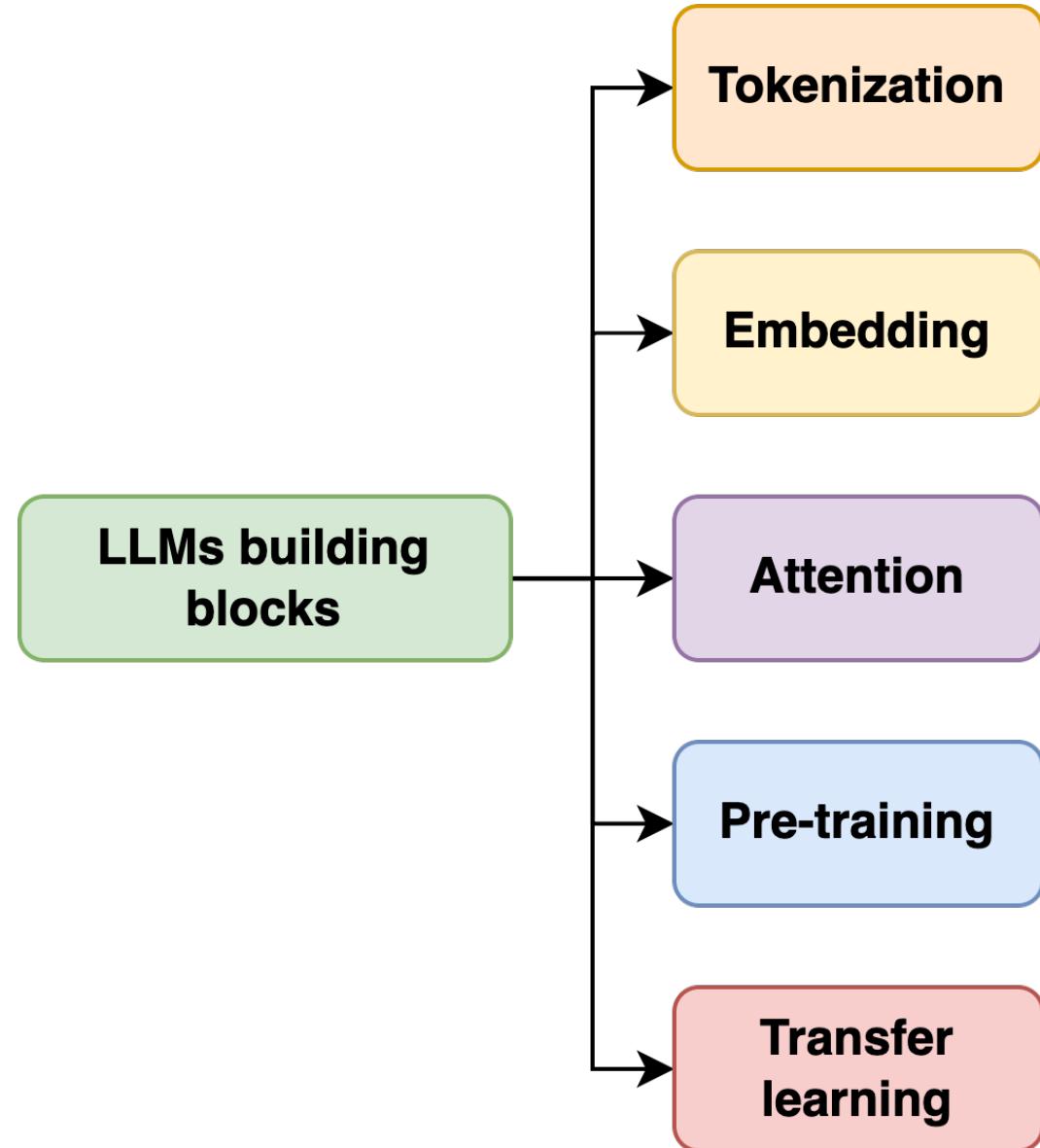
- Input embeddings
- Positional encoding
- Encoder:
 - Self-attention mechanism
 - Feed-forward neural network
- Decoder layers
- Multi-head attention
- Layer normalization
- Output layers



Architecture of LLMs

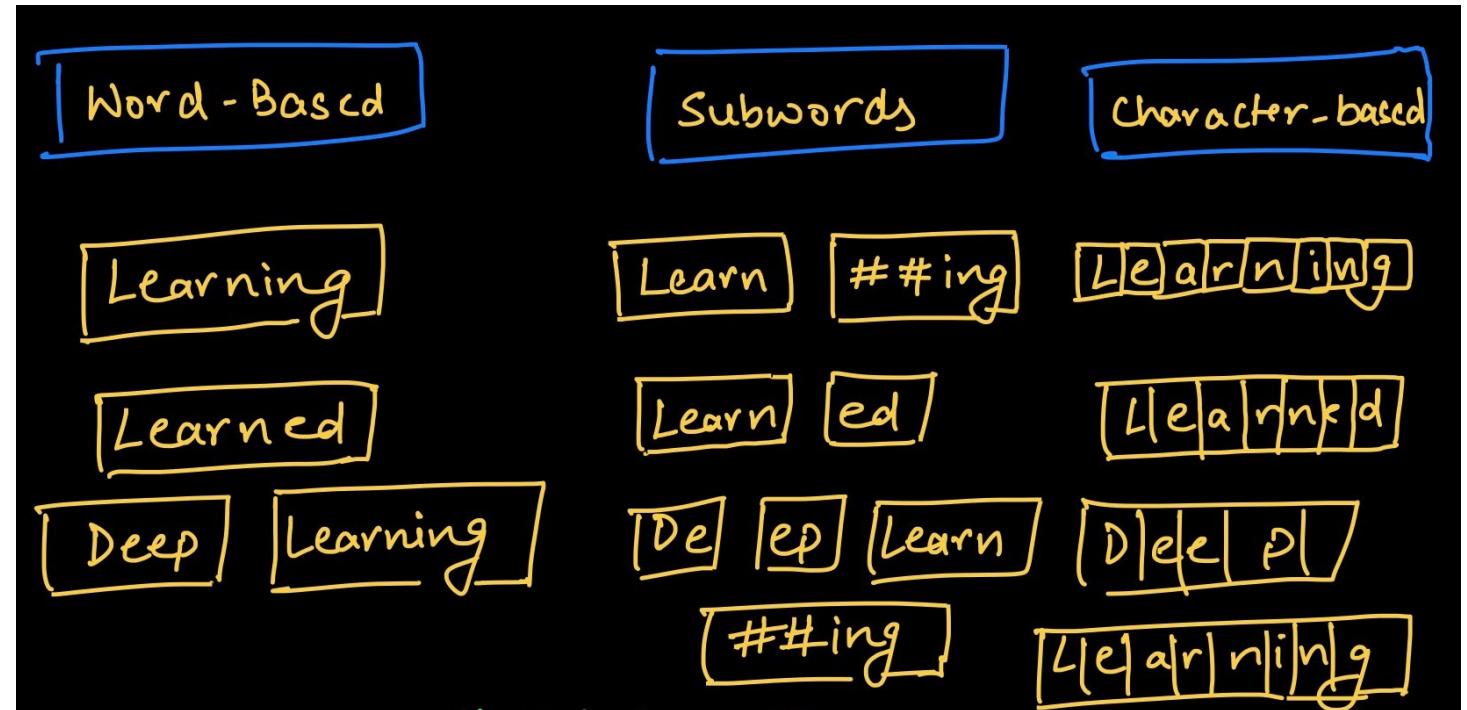
- The original Transformer architecture uses an encoder-decoder architecture, although variants with only an encoder or only a decoder also exist.

Architecture	Key Features	Notable LLMs	Use Cases
Encoder-only	<ul style="list-style-type: none">Captures bidirectional contextSuitable for NLU	BERT and other BERT-based architectures (RoBERTa, XLNet)	Text classification, question answering
Decoder-only	<ul style="list-style-type: none">Unidirectional language modelAutoregressive generation	GPT, PaLM	Text generation, text completion
Encoder-decoder	<ul style="list-style-type: none">Input text to target textAny text-to-text task	T5, BART	Summarization, translation, question answering, document classification



Tokenization

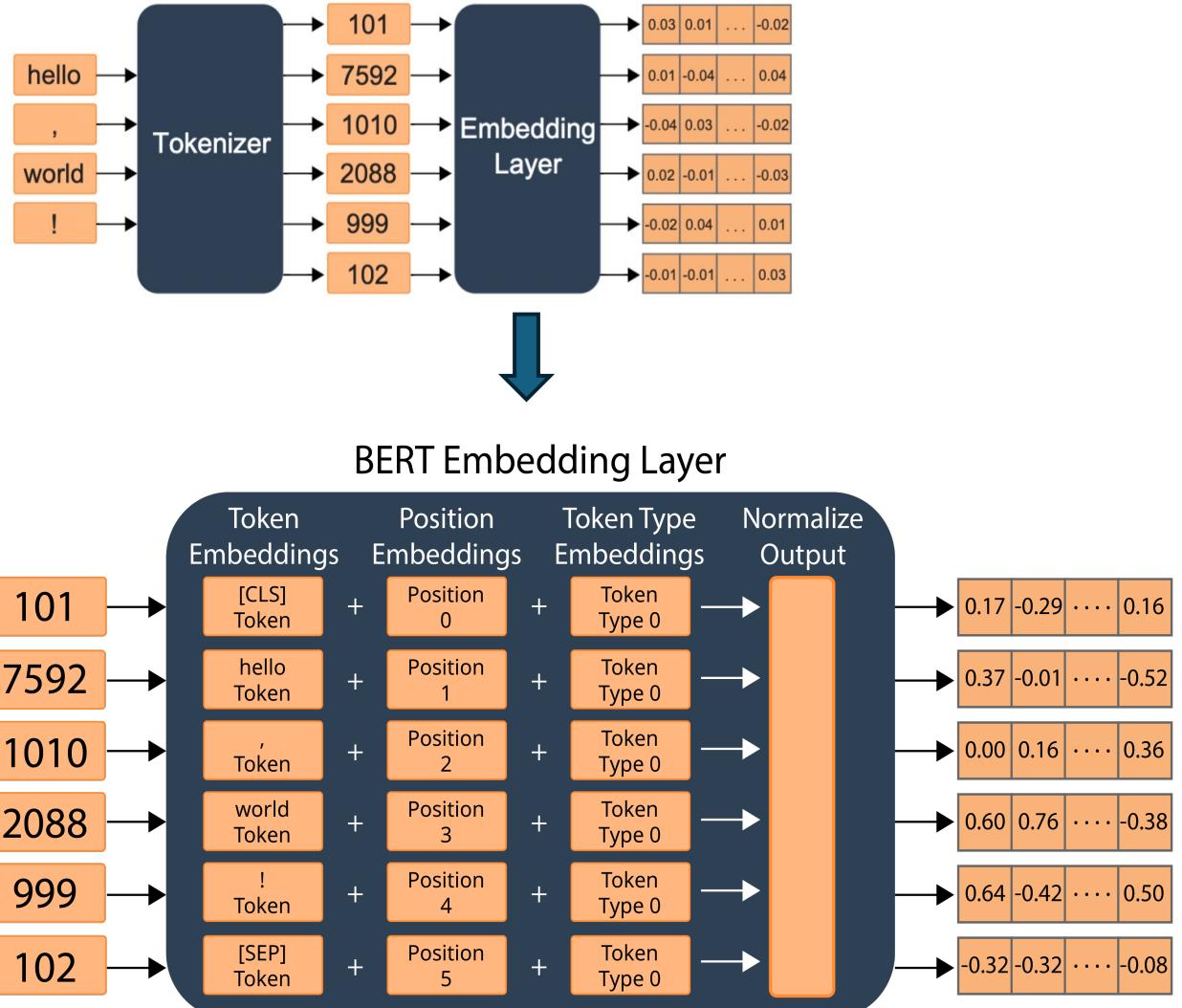
- Converting a sequence of text into tokens using subword algorithms like Byte Pair Encoding (BPE) or WordPiece.



<https://dswharshit.substack.com/p/the-evolution-of-tokenization-byte>

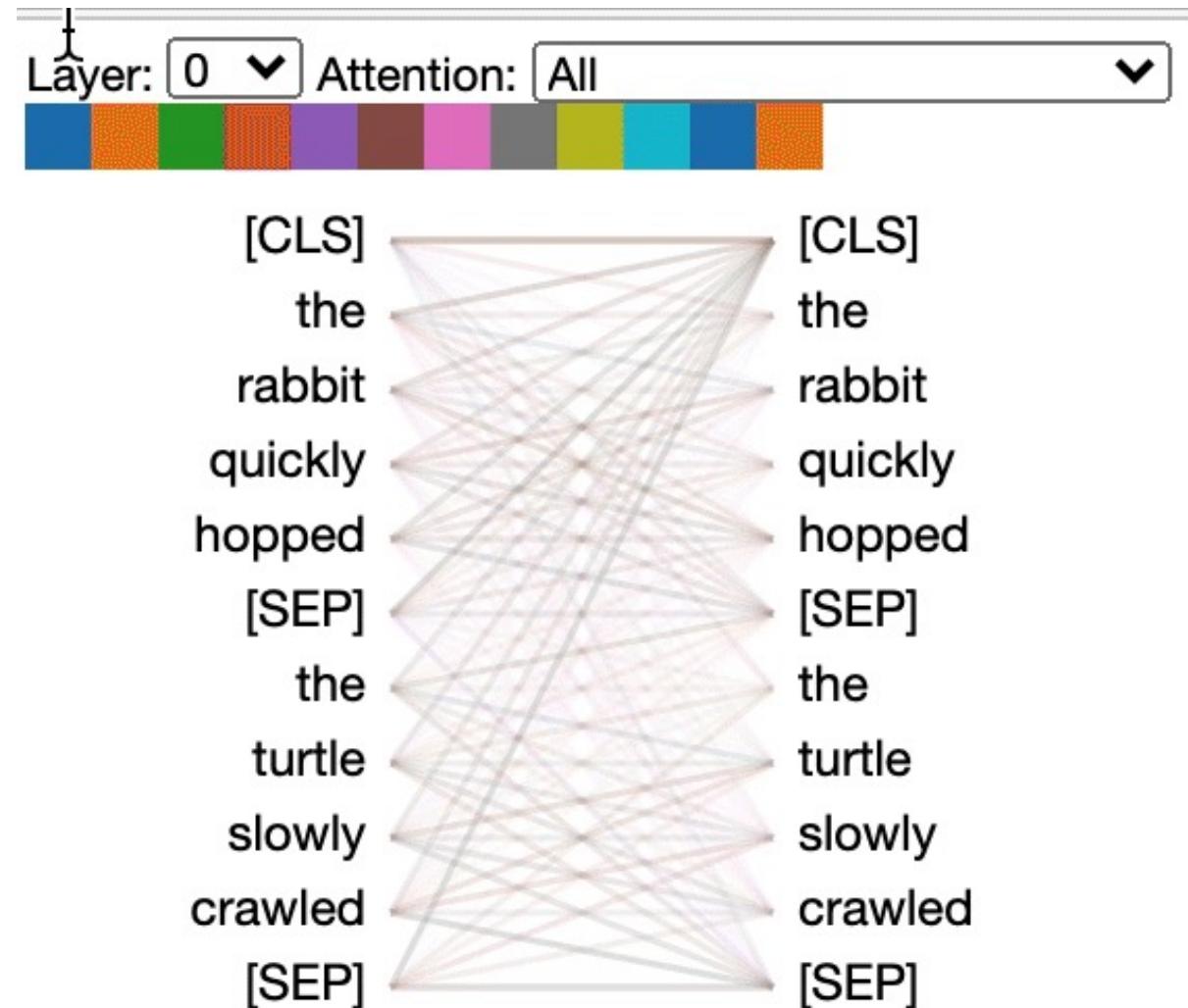
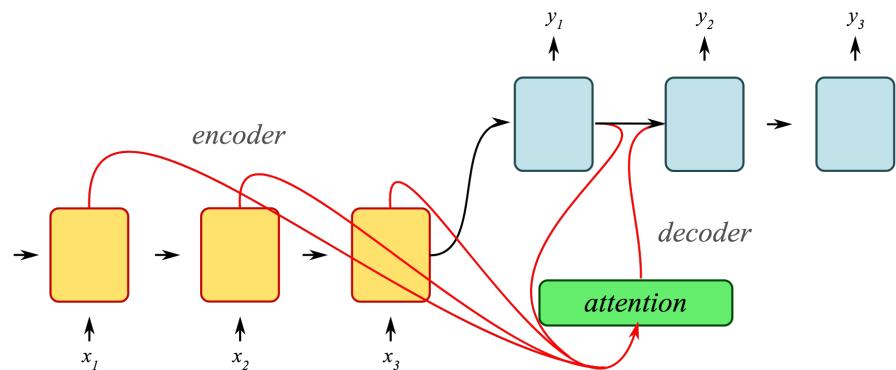
Embedding

- Embeddings are vector representations that map words or tokens into a multidimensional space, capturing semantic meanings.



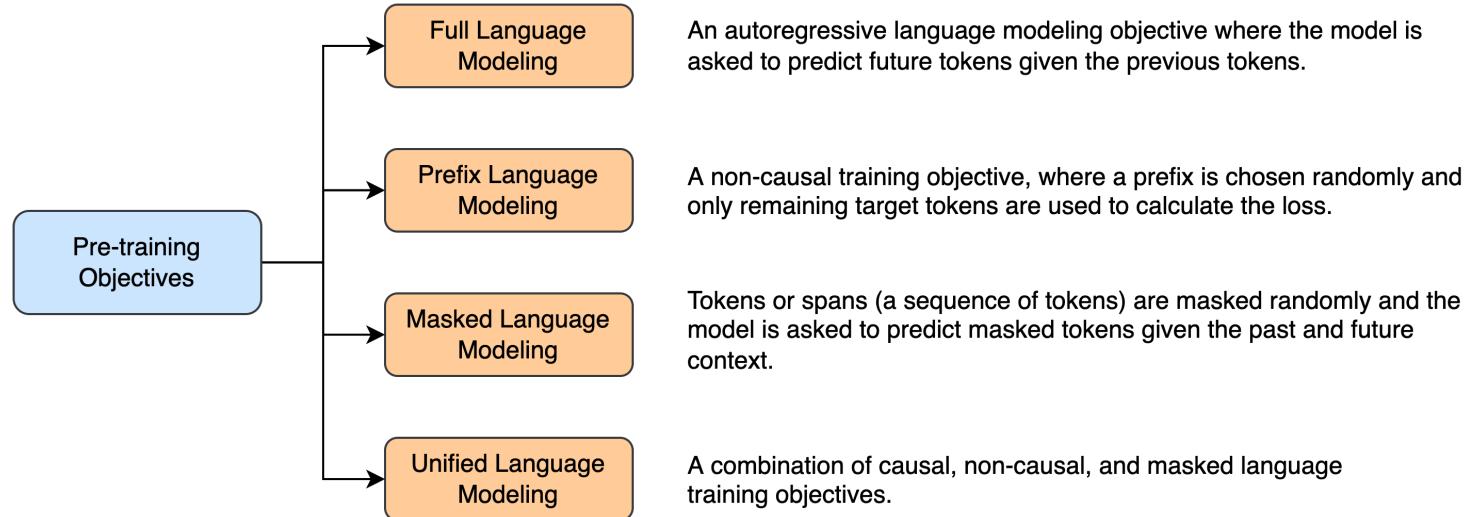
Attention

- The self-attention enables the model to weigh the significance of different elements within a given context.



Pre-training

- It is the initial training phase of an LLM on a large dataset to learn general language patterns before fine-tuning for specific tasks.



Full Language Modeling

May the force be with you

Prefix Language Modeling

May the force be with you

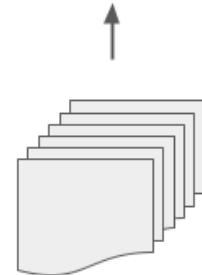
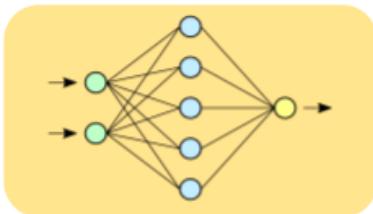
Masked Language Modeling

May the force be with you

Transfer Learning

- Transfer learning involves applying the knowledge obtained during pre-training to the new task on a smaller specific dataset.

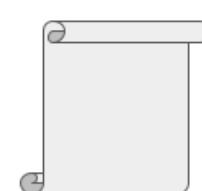
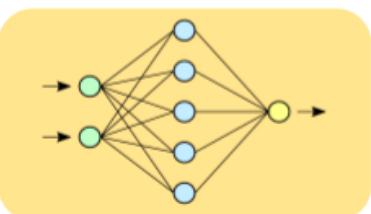
Pre-trained Model



Generic data

Transfer
Learning

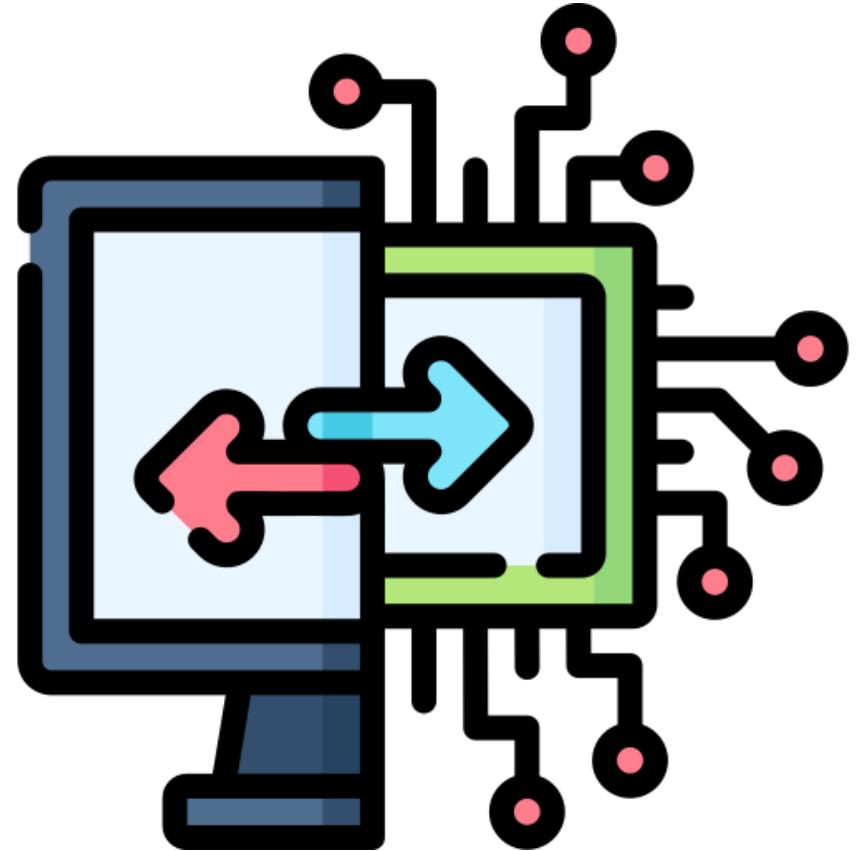
Fine-Tuned Model



Domain or task
specific data

Transfer Learning Techniques

- **Fine-tuning**
 - A process of adapting the pre-trained model to a new task or domain.
- **Zero-shot**
 - A technique wherein a model can perform a task without any labeled examples or training data for that specific task.
- **One-Shot**
 - A technique involving the training of a model with only one example or a very small number of examples per class.
- **Few-shot**
 - A technique referring to the training of a model with a small number of examples per class, typically more than one but fewer than what is required for conventional supervised learning.



Text Analysis

Text Analysis

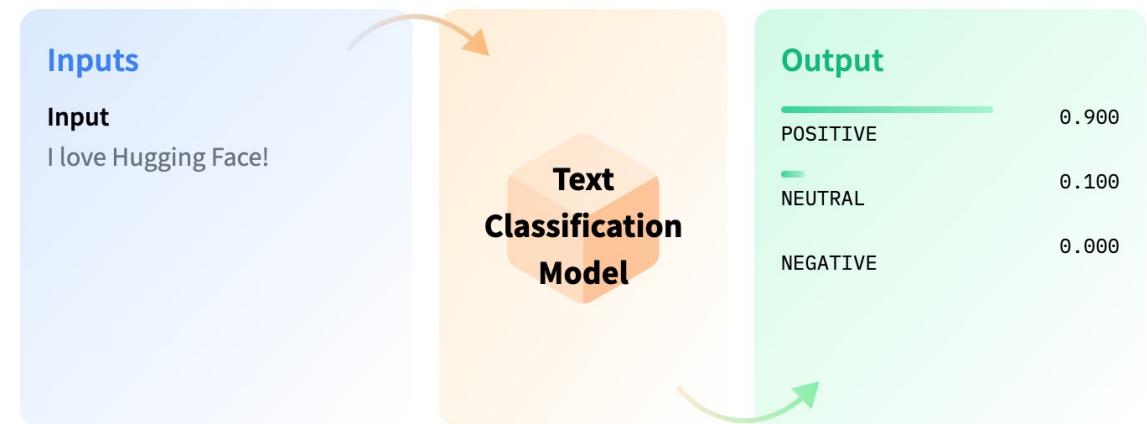
- Text analysis is the process of extracting meaningful insights, patterns, and information from textual data.
- It involves applying various techniques and methodologies to understand, interpret, and derive actionable insights from large volumes of text.



LLMs Use Cases for Text Analysis

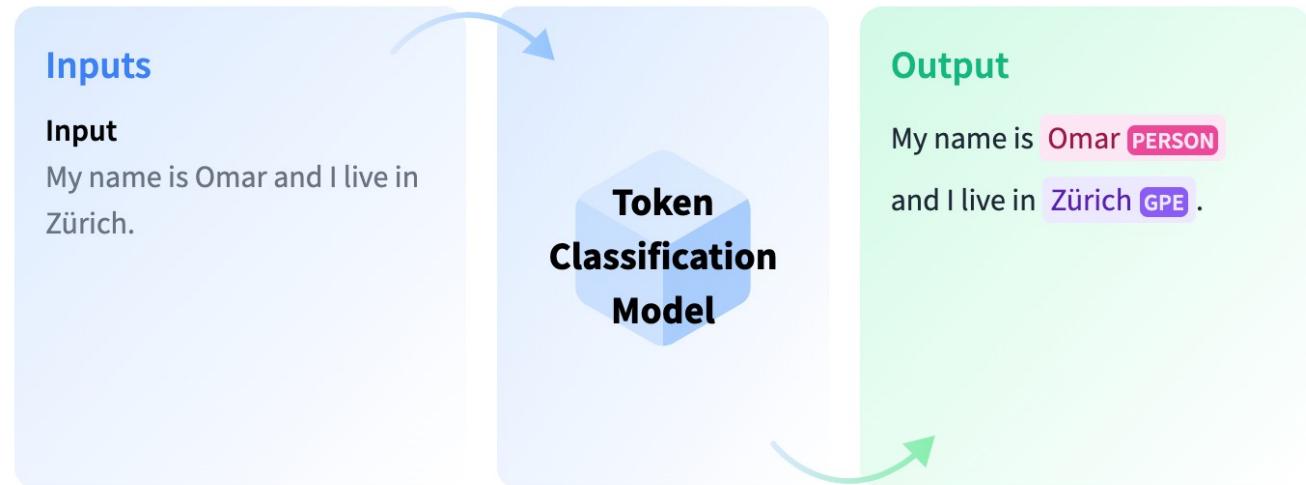
Text Classification

- Text classification is the task of assigning a label or class to a given text.
- Use cases:
 - sentiment analysis
 - spam detection
 - topic detection
 - language detection
 - offensive language identification



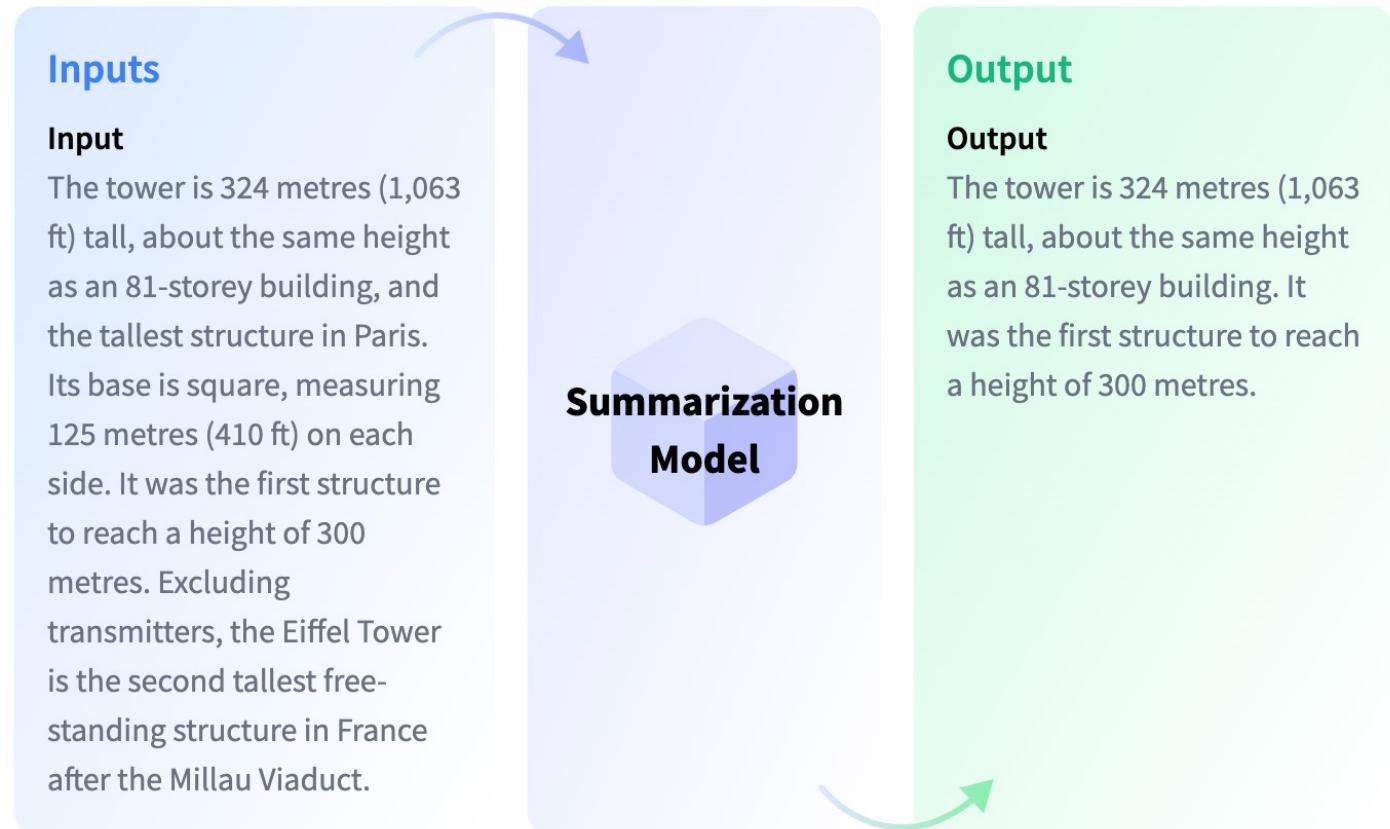
Token Classification

- Token classification is a natural language understanding task in which a label is assigned to some tokens in a text.
- Use cases:
 - Named Entity Recognition (NER)
 - Part-of-Speech (PoS) tagging
 - Word level language identification



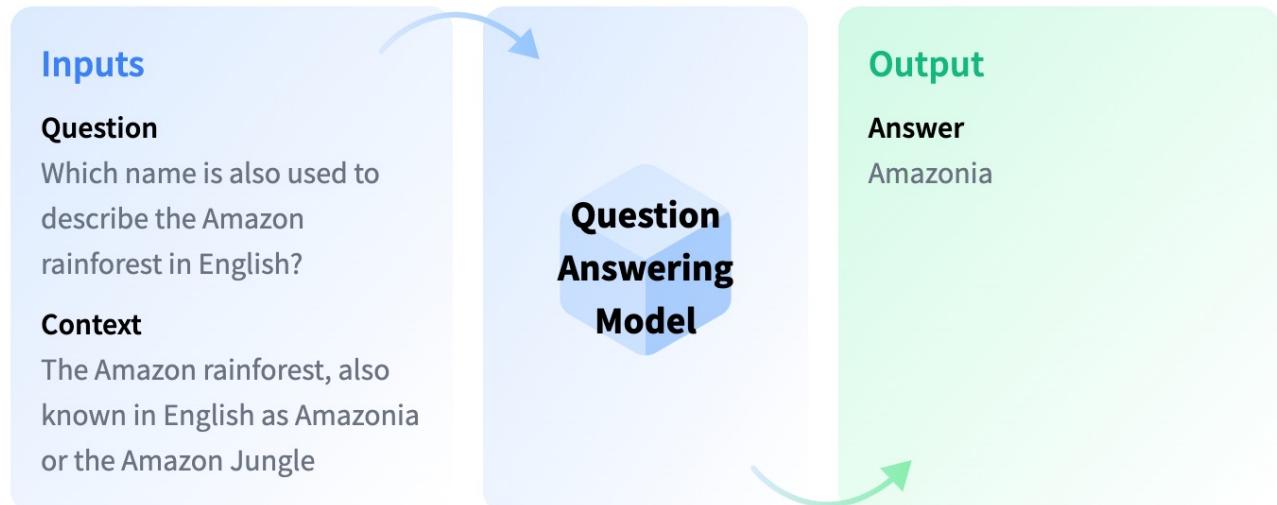
Text Summarization

- Summarization is the task of producing a shorter version of a document while preserving its important information.



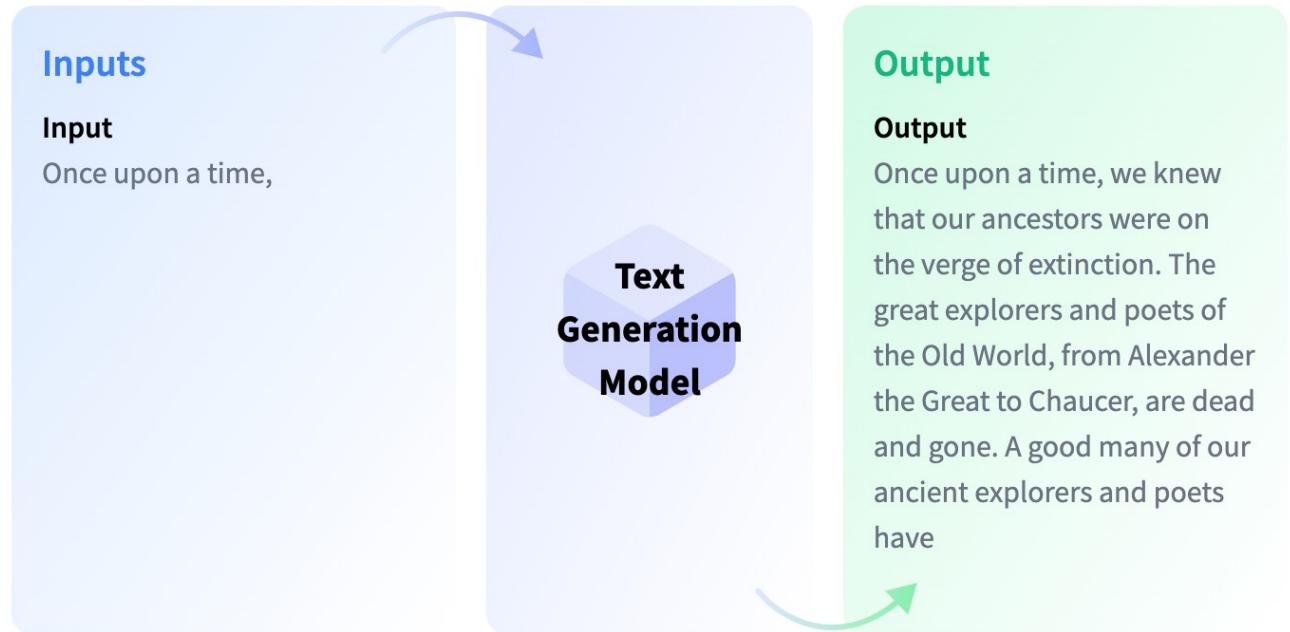
Question Answering

- Question Answering models can retrieve the answer to a question from a given text, which is useful for searching for an answer in a document.



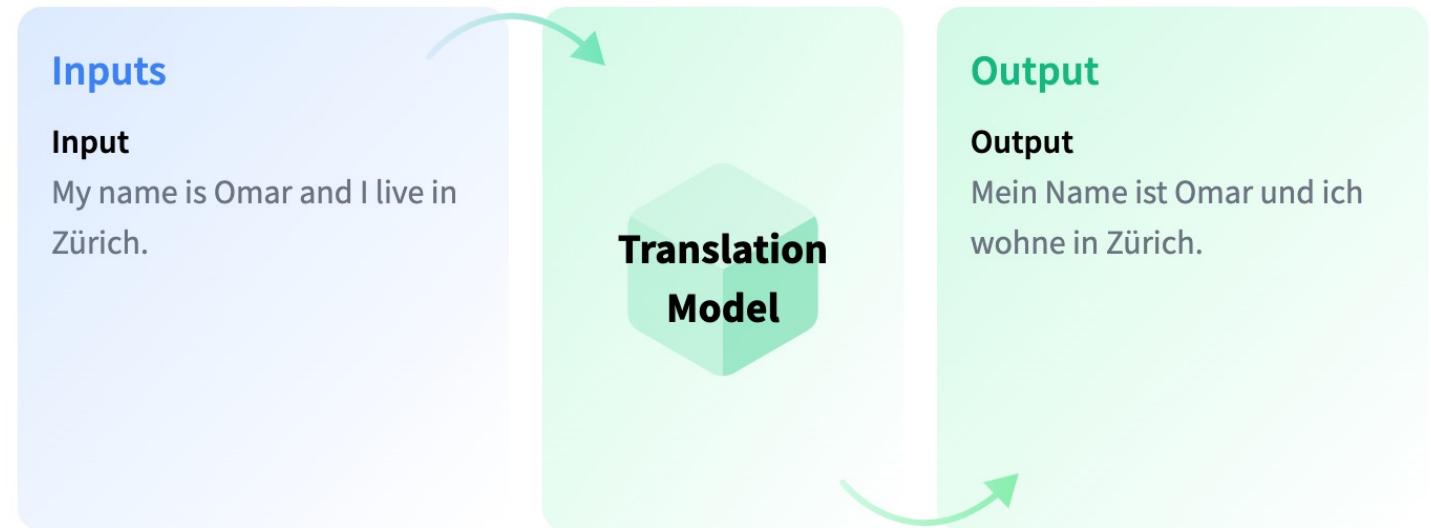
Text Generation

- Generating text is the task of generating new text given another text. These models can, for example, fill in incomplete text or paraphrase.



Translation

- Translation is the task of converting text from one language to another.



Case Study 1

Sentiment Analysis by Fine-tuning

Tools

- Kaggle with GPU
- Hugging Face

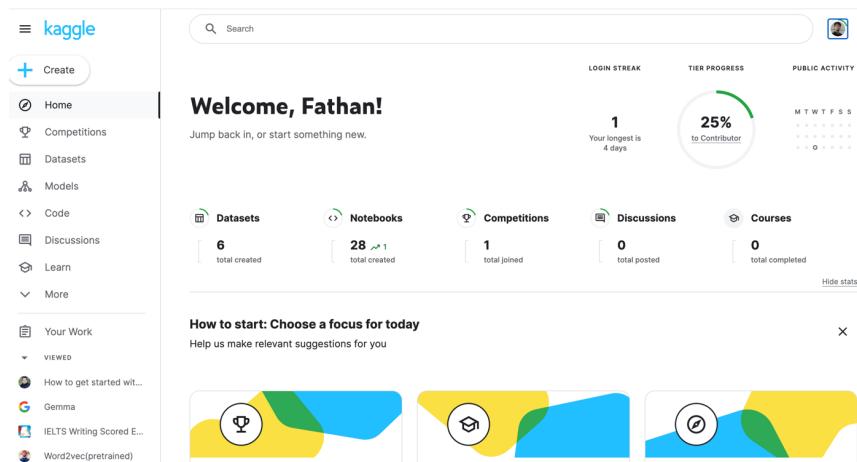
kaggle +



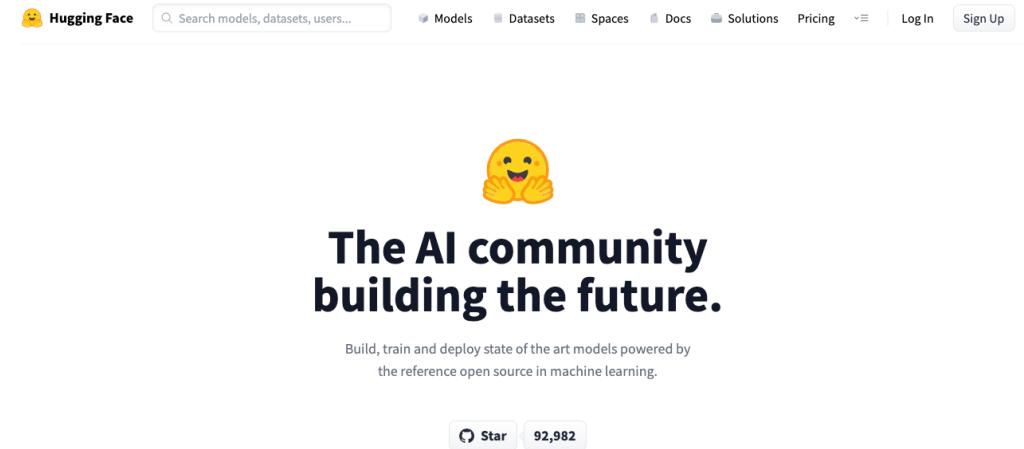
HUGGING FACE

Prerequisite

- Make sure that you already have a Kaggle and Hugging Face account!



<https://www.kaggle.com/>



<https://huggingface.co/>

Get the code here...

- <https://www.kaggle.com/code/fathanick/01-sentiment-analysis>

The screenshot shows a Jupyter Notebook interface on Kaggle. The title bar says "kaggle.com/code/fathanick/01-sentiment-analysis/edit". The notebook has a single code cell containing Python code for loading a dataset and generating train, validation, and test splits. The output shows the dataset being downloaded and split into three parts. Below the code cell, there are two buttons: "+ Code" and "+ Markdown". There are two additional code cells below, labeled [3] and [4], which show sample data and code for defining labels.

```
from datasets import load_dataset
dataset = load_dataset("sepidmnorozy/Indonesian_sentiment")

Downloading data: 100%|██████████| 1.61M/1.61M [00:00<00:00, 12.5MB/s]
Downloading data: 100%|██████████| 231k/231k [00:00<00:00, 4.14MB/s]
Downloading data: 100%|██████████| 456k/456k [00:00<00:00, 6.92MB/s]

Generating train split: 7926/0 [00:00<00:00, 138949.37 examples/s]
Generating validation split: 1132/0 [00:00<00:00, 47879.80 examples/s]
Generating test split: 2266/0 [00:00<00:00, 93746.42 examples/s]

[3]:
# see the sample
dataset["train"][10]

[3]: {'label': 0,
'text': ' suasana nya asik , romantis , dan ekslusif . pemandangan nya juga sangat bagus . tapi sayang , jalan untuk menuju ke sana nya masih rusak dan jauh .'}

[4]:
# define the labels

tags = ["NEGATIVE", "POSITIVE"]
id2label = {0: "NEGATIVE", 1: "POSITIVE"}
label2id = {"NEGATIVE": 0, "POSITIVE": 1}
num_labels = 2
```

Dataset

Hugging Face Search models, datasets, users...

Models Datasets

Datasets: sepidmnorozy/Indonesian_sentiment like 1

Tags: Croissant

Dataset card Viewer Files and versions Community

Dataset Viewer Auto-converted to Parquet API View in Dataset Viewer

Split (3)
train · 7.93k rows

Search this dataset

label	text
int64	string · lengths
0 1	3 567
1	bubur ayam yang lumayan rekomendasi di sekitaran bandung , tempat nya strategis mudah dicari , harga nya tidak merogoh kantong , tempat nya selalu ramai...
1	menu bebek relatif jarang di bandung dan bebek garang ini salah satu yang terenak , rasa bebek nya enak dan meresap ke daging nya , daging nya lembut dan...
1	hampir lebih 5 kali saya ke sini . dim sum nya yahud dan harga nya terjangkau banget . tempat nya asyik dan sangat nyaman . pelayan nya ramah walaupun kita...
1	tempat nya dekat dengan factory outlet jadi habis berbelanja kita bisa mampir makan ke tempat ini . menu nya banyak sekali mulai dari lokal dan luar negri ...
1	saya tidak sengaja menemuka warung bakso dan batagor di jalan serayu ini saat melewati jalan brantas yang ada di samping jalan serayu . saat itu saya...
0	makanan di rs selalu tidak enak

https://huggingface.co/datasets/sepidmnorozy/Indonesian_sentiment

Fine-tuning with IndoBERTweet

- In this tutorial, we're going to fine-tune IndoBERTweet model sentiment data.
- IndoBERTweet is the first large-scale pretrained model for Indonesian Twitter that is trained by extending a monolingually trained Indonesian BERT model with additive domain-specific vocabulary.

INDOBERTWEET: A Pretrained Language Model for Indonesian Twitter with Effective Domain-Specific Vocabulary Initialization

Fajri Koto Jey Han Lau Timothy Baldwin

School of Computing and Information Systems
The University of Melbourne

ffajri@student.unimelb.edu.au, jeyhan.lau@gmail.com, tb@lwdwin.net

Abstract

We present INDOBERTWEET, the first large-scale pretrained model for Indonesian Twitter that is trained by extending a monolingually-trained Indonesian BERT model with additive domain-specific vocabulary. We focus in particular on efficient model adaptation under vocabulary mismatch, and benchmark different ways of initializing the BERT embedding layer for new word types. We find that initializing with the average BERT subword embedding makes pretraining five times faster, and is more effective than proposed methods for vocabulary adaptation in terms of extrinsic evaluation over seven Twitter-based datasets.¹

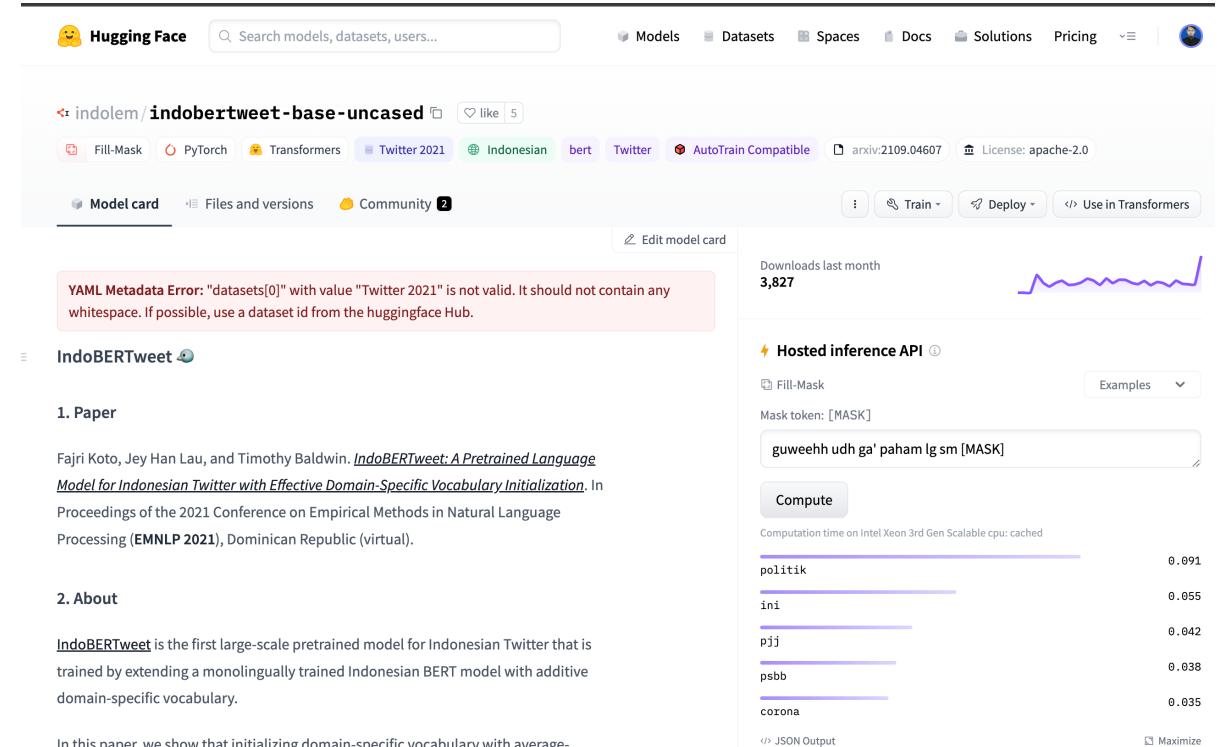
1 Introduction

Transformer-based pretrained language models (Vaswani et al., 2017; Devlin et al., 2019; Liu et al., 2019; Radford et al., 2019) have become the back-

term in biology. To tackle this problem, Poerner et al. (2020); Tai et al. (2020) proposed simple methods to domain-extend the BERT vocabulary: Poerner et al. (2020) initialize new vocabulary using a learned projection from word2vec (Mikolov et al., 2013), while Tai et al. (2020) use random initialization with weight augmentation, substantially increasing the number of model parameters.

New vocabulary augmentation has been also conducted for language-adaptive pretraining, mainly based on multilingual BERT (mBERT). For instance, Chau et al. (2020) replace 99 “unused” WordPiece tokens of mBERT with new common tokens in the target language, while Wang et al. (2020) extend mBERT vocabulary with non-overlapping tokens ($[\mathbb{V}_{\text{MBERT}} - \mathbb{V}_{\text{new}}]$). These two approaches use random initialization for new WordPiece token embeddings.

In this paper, we focus on the task of learning



<https://huggingface.co/indolem/indobertweet-base-uncased>

GPU Activation

01_sentiment_analysis Draft saved

File Edit View Run Add-ons Help

Code ▾ Draft Session off (run a cell to start) ⚡ ⏪ ⏹ ⋮

```
+ ✎ ⟲ ⟳ | ⏪ ⏹ | ⏴ ⏵ Run All | Code ▾
```

```
from datasets import load_dataset

dataset = load_dataset("sepidmnorozy/Indonesian_sentiment")
```

Downloading data: 100%|██████████| 1.61M/1.61M [00:00<00:00, 12.5MB/s]
Downloading data: 100%|██████████| 231k/231k [00:00<00:00, 4.14MB/s]
Downloading data: 100%|██████████| 456k/456k [00:00<00:00, 6.92MB/s]

Generating train split: 7926/0 [00:00<00:00, 138949.37 examples/s]

Generating validation split: 1132/0 [00:00<00:00, 47879.80 examples/s]

Generating test split: 2266/0 [00:00<00:00, 93746.42 examples/s]

+ Code + Markdown

[3]: # see the sample
dataset["train"][10]

Share ⌂ Save Version 1

Output ^
/kaggle/working ↗

Session options ^

ACCELERATOR ▾ GPU T4 ×2

None

GPU T4 x2

GPU P100

TPU VM v3-8

ENVIRONMENT ▾ Pin to original environment (2024-02-28)

Install Packages

```
# Install required packages
!pip install transformers[torch]
!pip install -U datasets
!pip install huggingface_hub
!pip install accelerate -U
```

Connect to Hugging Face

- Get access token from your Hugging Face account
 - Go to: <https://huggingface.co/settings/tokens>

The screenshot shows the Hugging Face user interface. At the top, there's a navigation bar with the Hugging Face logo, a search bar, and links for Models, Datasets, and Spaces. Below the navigation is a sidebar with links for Profile, Account, Organizations, Billing, Access Tokens (which is highlighted), SSH and GPG Keys, Webhooks, and Panore. The main content area is titled "Access Tokens" and contains a sub-section "User Access Tokens". It explains what access tokens are and how they work. A table lists an existing token for "hanabi" with "WRITE" permissions. There are "Manage", "Show", and "Copy" buttons next to the token. A "New token" button is at the bottom.



The screenshot shows a Jupyter Notebook login dialog. It features a yellow emoji of a smiling face with hands clasped together. Below the emoji is the text "Copy a token from [your Hugging Face tokens page](#) and paste it below." Further down, it says "Immediately click login after copying your token or it might be stored in plain text in this notebook file." A text input field contains a redacted token, followed by a "Token:" label. A checked checkbox labeled "Add token as git credential?" is present. At the bottom is a "Login" button.

Load Dataset

```
→ from datasets import load_dataset  
  
dataset = load_dataset("sepidmnorozy/Indonesian_sentiment")  
  
Downloading data: 100%|██████████| 1.61M/1.61M [00:00<00:00, 12.5MB/s]  
Downloading data: 100%|██████████| 231k/231k [00:00<00:00, 4.14MB/s]  
Downloading data: 100%|██████████| 456k/456k [00:00<00:00, 6.92MB/s]  
  
Generating train split: 7926/0 [00:00<00:00, 138949.37 examples/s]  
  
Generating validation split: 1132/0 [00:00<00:00, 47879.80 examples/s]  
  
Generating test split: 2266/0 [00:00<00:00, 93746.42 examples/s]  
  
+ Code + Markdown  
  
[3]:  
    # see the sample  
    dataset["train"][10]  
  
[3]: {'label': 0,  
      'text': " suasana nya asik , romantis , dan ekslusif . pemandangan nya juga sangat bagus . tapi sayang , jalan untuk menuju ke sana nya masih rusak dan jauh ."}  
  
[4]:  
    # define the labels  
  
    tags = ["NEGATIVE", "POSITIVE"]  
    id2label = {0: "NEGATIVE", 1: "POSITIVE"}  
    label2id = {"NEGATIVE": 0, "POSITIVE": 1}  
    num_labels = 2
```

Load Packages

```
from transformers import (
    AutoTokenizer,
    AutoModelForSequenceClassification,
    DataCollatorWithPadding,
    TrainingArguments,
    Trainer,
)
```

Initialize The Model

```
def model_init(model_name):
    global tokenizer
    global data_collator
    global tr_model

    model = AutoModelForSequenceClassification.from_pretrained(
        model_name,
        num_labels=num_labels,
        id2label=id2label,
        label2id=label2id
    )
    tokenizer = AutoTokenizer.from_pretrained(model_name, model_max_length=max_length)

    return model, tokenizer
```

Tokenization and Preprocessing

```
def tokenize_function(examples):
    # process the input sequence
    tokenized_input = tokenizer(examples["text"],
                                truncation=True,
                                padding="max_length",
                                max_length=max_length)

    return tokenized_input

def preprocessing():
    tokenized_train_data = dataset["train"].map(tokenize_function, batched=True)
    tokenized_val_data = dataset["validation"].map(tokenize_function, batched=True)
    tokenized_test_data = dataset["test"].map(tokenize_function, batched=True)

    return tokenized_train_data, tokenized_val_data, tokenized_test_data
```

Define Function for Evaluation

```
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score, classification_report
import numpy as np

def compute_metrics(p):
    pred, labels = p
    pred = np.argmax(pred, axis=1)

    true_labels = [tags[1] for l in labels]
    true_predictions = [tags[pr] for pr in pred]

    report = classification_report(true_labels, true_predictions, digits=4)
    acc = accuracy_score(y_true=true_labels, y_pred=true_predictions)
    rec = recall_score(y_true=true_labels, y_pred=true_predictions, average="macro")
    prec = precision_score(y_true=true_labels, y_pred=true_predictions, average="macro")
    f1 = f1_score(y_true=true_labels, y_pred=true_predictions, average="macro", zero_division=1.0)

    print("Classification Report:\n{}\n".format(report))
    return {"accuracy": acc, "precision": prec, "recall": rec, "f1": f1}
```

Function for Generating Confusion Matrix

```
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

def generate_confusion_matrix(true_labels, pred_labels, num_labels):
    cm = confusion_matrix(true_labels, pred_labels)
    labels = [id2label[i] for i in range(num_labels)]
    plt.figure(figsize=(8, 6))
    sns.set(font_scale=1.5)
    sns.heatmap(cm,
                annot=True,
                fmt="d",
                cmap="Blues",
                xticklabels=labels,
                yticklabels=labels)
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    plt.yticks(rotation=0)
    plt.show()
```

Train The Model

```
def train_model(model_name, output_dir, learning_rate, num_epochs):  
  
    model, tokenizer = model_init(model_name)  
    train_tokenized, val_tokenized, test_tokenized = preprocessing()  
  
    training_args = TrainingArguments(  
        output_dir=output_dir,  
        logging_strategy="epoch",  
        evaluation_strategy="epoch",  
        save_strategy="epoch",  
        save_total_limit = 1,  
        learning_rate=learning_rate,  
        num_train_epochs=num_epochs,  
        per_device_train_batch_size=32,  
        per_device_eval_batch_size=32,  
        load_best_model_at_end=True,  
        push_to_hub=True, # to push to hub during the training  
    )  
  
    trainer = Trainer(  
        model=model,  
        args=training_args,  
        train_dataset=train_tokenized,  
        eval_dataset=val_tokenized,  
        tokenizer=tokenizer,  
        compute_metrics=compute_metrics,  
    )  
  
    trainer.train()  
    trainer.save_model(output_dir)  
    trainer.push_to_hub(commit_message="Training complete")  
  
    # Get the evaluation results  
    trainer.eval_dataset=test_tokenized  
    evaluation_results = trainer.evaluate()  
    print(evaluation_results)  
  
    # make prediction on the test set  
    predictions = trainer.predict(test_tokenized)  
    pred_labels = np.argmax(predictions.predictions, axis=1)  
    true_labels = test_tokenized["label"]  
  
    # Generate confusion matrix  
    generate_confusion_matrix(true_labels, pred_labels, num_labels)  
  
    # Show log history  
    log_history = pd.DataFrame(trainer.state.log_history)  
    log_history = log_history.fillna(0)  
    log_history = log_history.groupby(["epoch"]).sum()  
  
    log_history[["loss", "eval_loss"]].plot()  
    plt.show()
```

```
train_model(  
    model_name="indolem/indobertweet-base-uncased",  
    output_dir="indo_sentiment_indobertweet",  
    learning_rate=0.0001,  
    num_epochs=3  
)
```

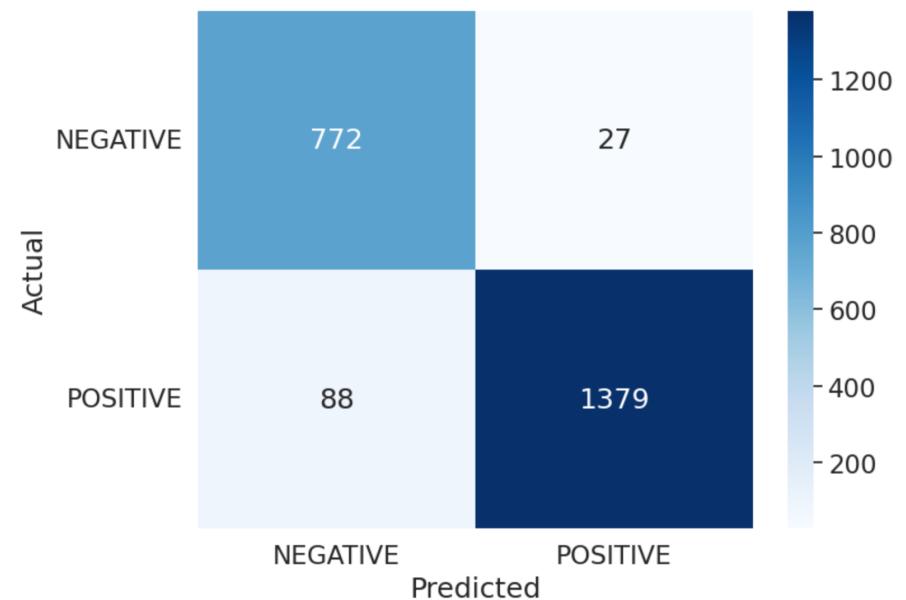
Results

[372/372 04:55, Epoch 3/3]

Epoch	Training Loss	Validation Loss	Accuracy	Precision	Recall	F1
1	0.183500	0.103949	0.960247	0.953081	0.962508	0.957393
2	0.063900	0.109426	0.963781	0.958891	0.963151	0.960946
3	0.017600	0.136030	0.965548	0.963370	0.961879	0.962616

Classification Report:

	precision	recall	f1-score	support
NEGATIVE	0.8977	0.9662	0.9307	799
POSITIVE	0.9808	0.9400	0.9600	1467
accuracy			0.9492	2266
macro avg	0.9392	0.9531	0.9453	2266
weighted avg	0.9515	0.9492	0.9496	2266



Hugging Face

fathan/indo_sentiment_indobertweet like 0

Text Classification Transformers TensorBoard Safetensors bert generated_from_trainer Inference Endpoints License: apache-2.0

Model card Files and versions Training metrics Community Settings Train Deploy Use in Transformers Edit model card

indo_sentiment_indobertweet

This model is a fine-tuned version of [indolem/indobertweet-base-uncased](#) on an unknown dataset. It achieves the following results on the evaluation set:

- Loss: 0.1360
- Accuracy: 0.9655
- Precision: 0.9634
- Recall: 0.9619
- F1: 0.9626

Model description

More information needed

Downloads last month 0

Safetensors Model size 111M params Tensor type F32

Inference API

Text Classification Examples

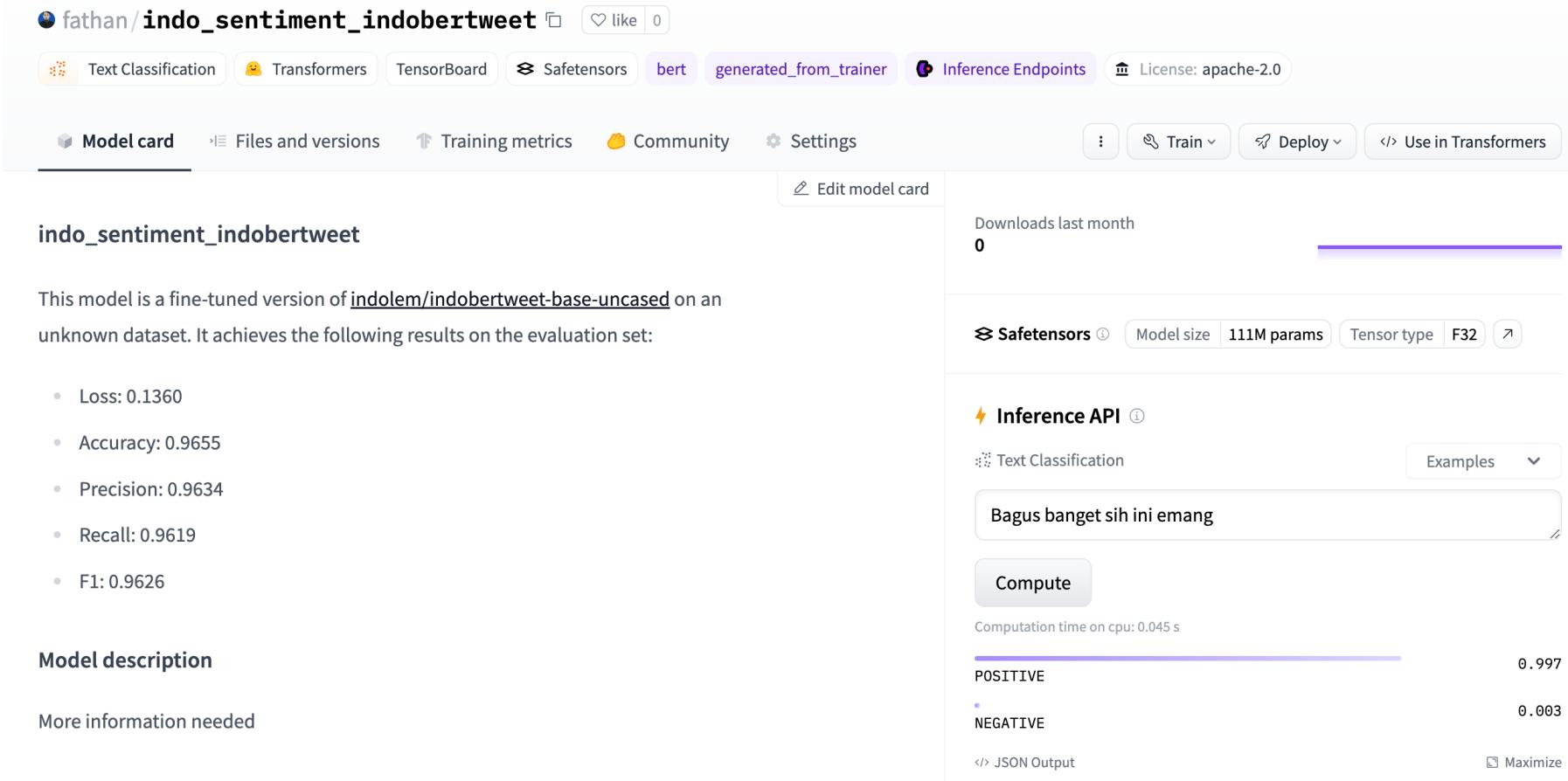
Bagus banget sih ini emang

Compute

Computation time on cpu: 0.045 s

Category	Score
POSITIVE	0.997
NEGATIVE	0.003

JSON Output Maximize



Case Study 2

Zero-shot classification: Topic Detection

Zero-shot Classification

- These models classify text into any categories we want by inputting them as labels.



⚡ Inference API ⓘ

⌘ Zero-Shot Classification Examples ▾

Apple umumkan harga iPhone 14.

Possible class names (comma-separated)

teknologi, olahraga, kuliner, bisnis

Allow multiple true classes

Compute

Computation time on cpu: 0.064 s

Output Category	Probability
teknologi	0.939
bisnis	0.755
olahraga	0.414
kuliner	0.236

↪ JSON Output Maximize

<https://huggingface.co/ilos-vigil/bigbird-small-indonesian-nli>

Define The Model

```
from transformers import pipeline
```

```
# define the pipeline
classifier = pipeline("zero-shot-classification", model="facebook/bart-large-mnli")
```

Prediction

```
# sample 1
candidate_labels_sentiment = ["positive", "negative"]

text1 = "I hate this movie and I don't recommend people to watch this."
classifier(sequences=text1,
           candidate_labels=candidate_labels_sentiment,
           multi_label=True)
```

+ Code

+ Markdown

```
# sample 2
text2 = "I really love this movie!"
classifier(sequences=text2,
           candidate_labels=candidate_labels_sentiment,
           multi_label=True)
```

```
# sample 3
candidate_labels_news = ["world", "sports", "business", "sci/tech"]

text3 = "Liverpool were not given a penalty by VAR in the final minute of their pulsating 1-1 draw against Manchester City on Super Sunday."
classifier(sequences=text3,
           candidate_labels=candidate_labels_news,
           multi_label=True)
```

```
# sample 4
text4 = "Microsoft has expanded the availability of its AI-powered cybersecurity assistant, Copilot for Security, using the power of generative AI (Gen AI)"
classifier(sequences=text4,
           candidate_labels=candidate_labels_news,
           multi_label=True)
```



thank you