TRANSFORMER FOR LANGUAGE TRANSLATION

- TUSHAR ARORA

POINTS TO ENSURE

- Making From Scratch
- Without Huggingface Transformer
- Any translation dataset
- Using either PyTorch or TensorFlow.
- Code is modular and easy to understand

TENSORFLOW VS PYROCH

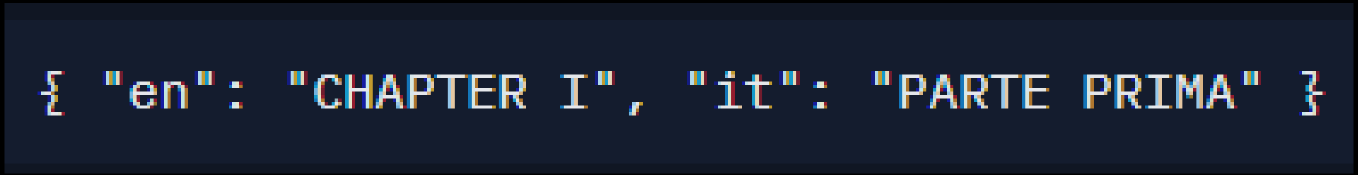
I CHOOSE PYTORCH BECAUSE OF

- Easy to use
- Community Support
- Flexible
- Deployment Using Torchserve
- Performance

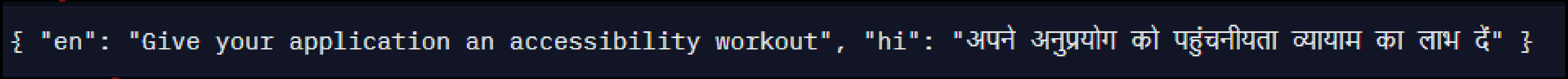
DATASET

- OPUS
- WMT
- IWSLT
- Multi30k
- TED Talk
- MultiUN

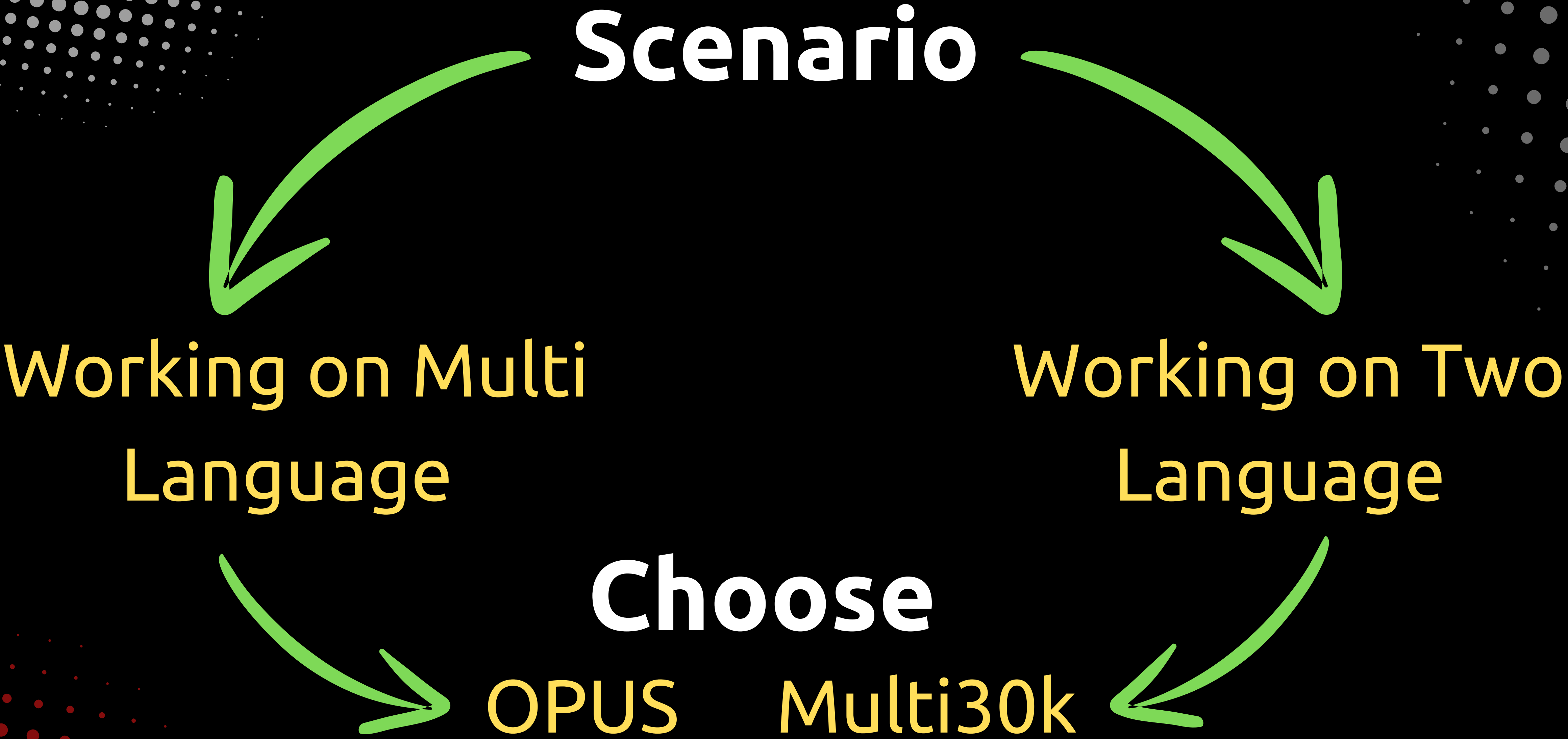
- Tatoebe
- Open subtitle
- Europarl
- UN Parallel
- Wikipedia Multilingual Corpus
- Cflit IIT Bombay



```
{ "en": "CHAPTER I", "it": "PARTE PRIMA" }
```



```
{ "en": "Give your application an accessibility workout", "hi": "अपने अनुप्रयोग को पहुंचनीयता व्यायाम का लाभ दें" }
```



Scenario

Working on Multi Language

Working on Two Language

Choose

OPUS

Multi30k

Single Language Translation

OPUS Book Dataset

- There are 16 Language combinations with different rows
- Ex : English to Portuguese(1.4k) - Result will not Good because of low data
- Ex : English to French(127k) - It will takes too much time
- Ex : English to Italian(32.3k) - It can

Multi Language Translation

Single Language Translation

English to Italian

Single Language Translation

English to Hindi

Framework like Streamlit

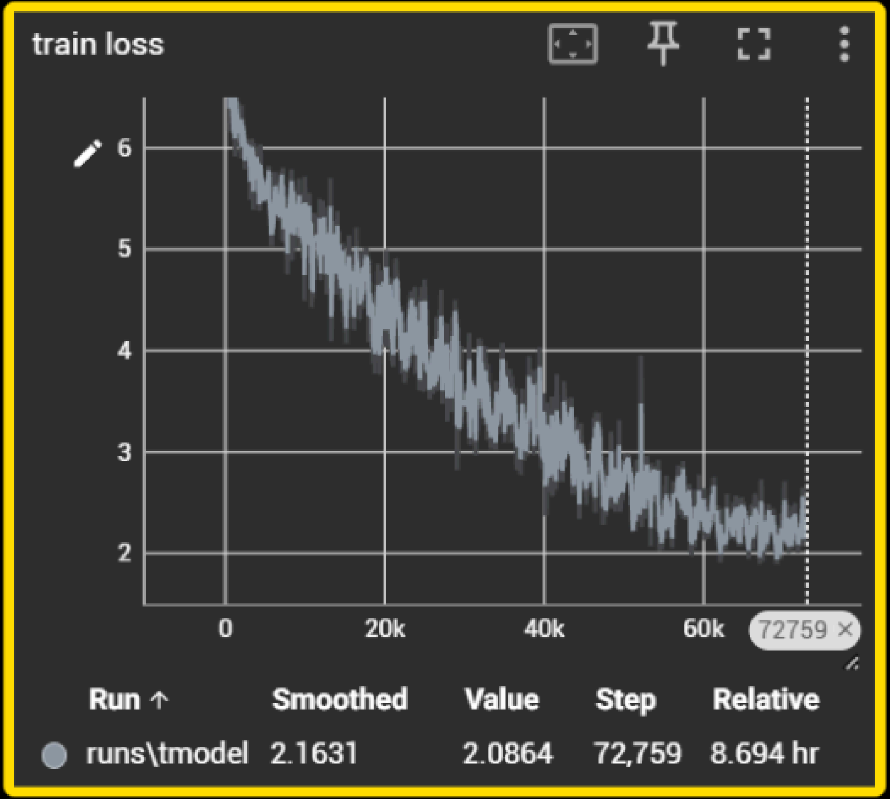
Without Modular Coding
(English to Italian)
OPUS dataset

With
Modular Coding
(English to Italian)
OPUS dataset

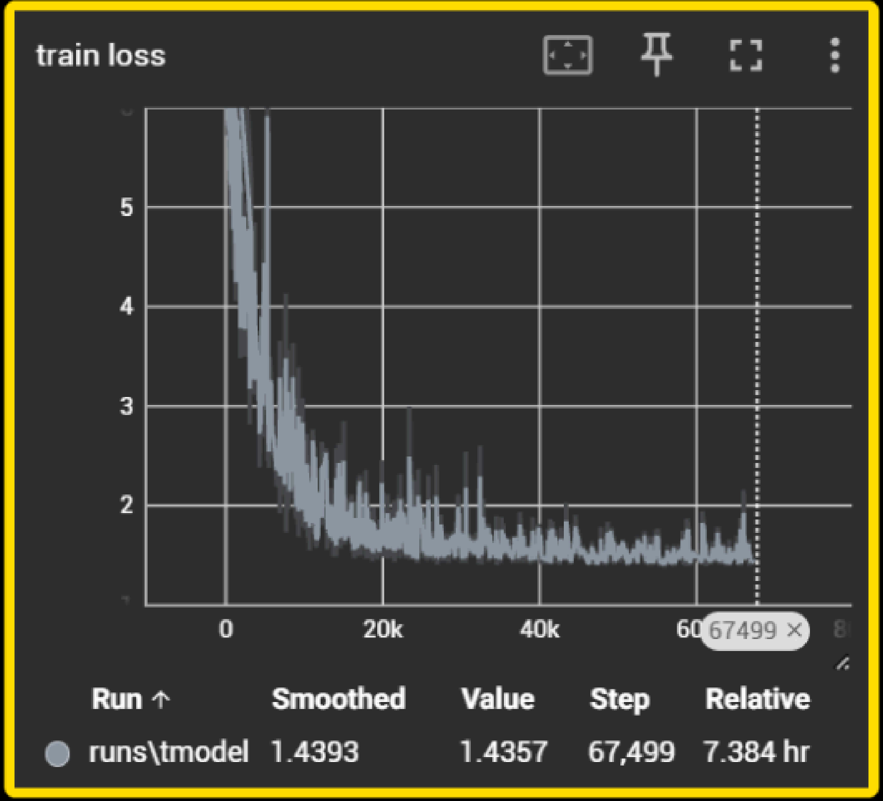
With
Modular Coding
(English to Hindi)
IIT Bombay Dataset

(English to Italian)
OPUS dataset
32k with 20 epoch

(English to Hindi)
IIT Bombay Dataset
30k with 20 epoch

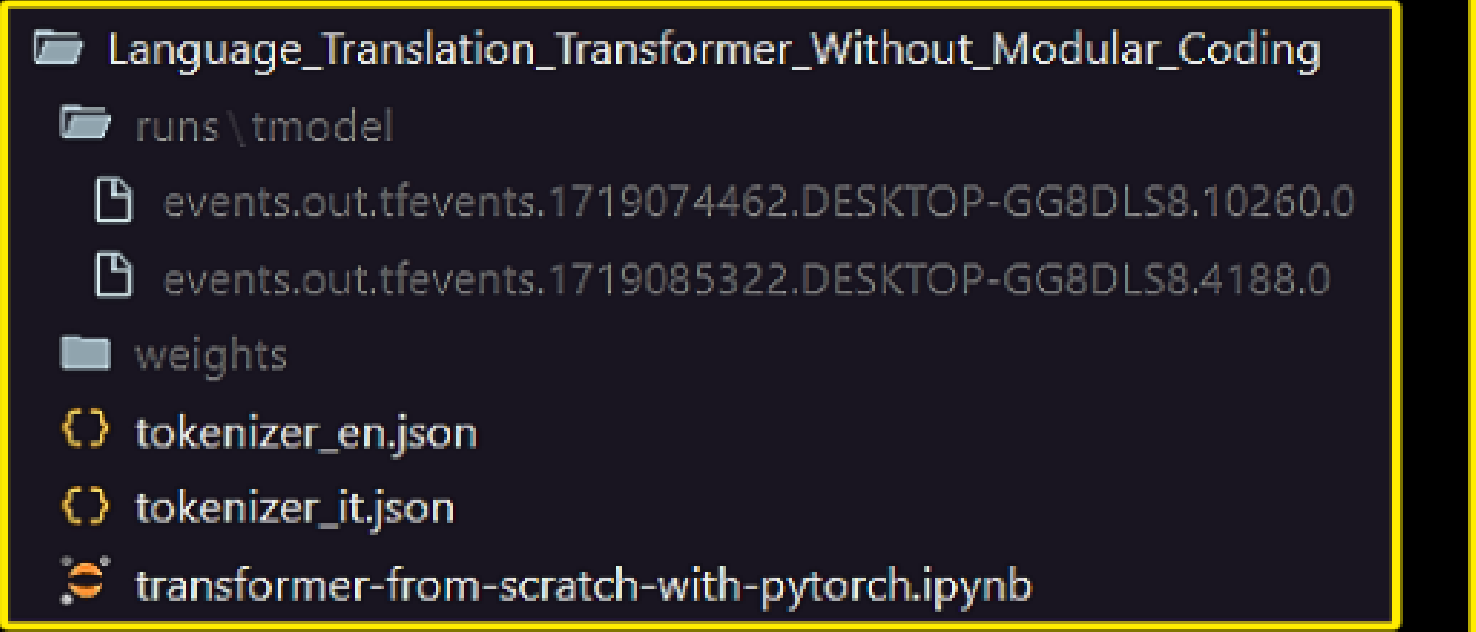


Run ↑	Smoothed	Value	Step	Relative
● runs\tmodel	2.1631	2.0864	72,759	8.694 hr

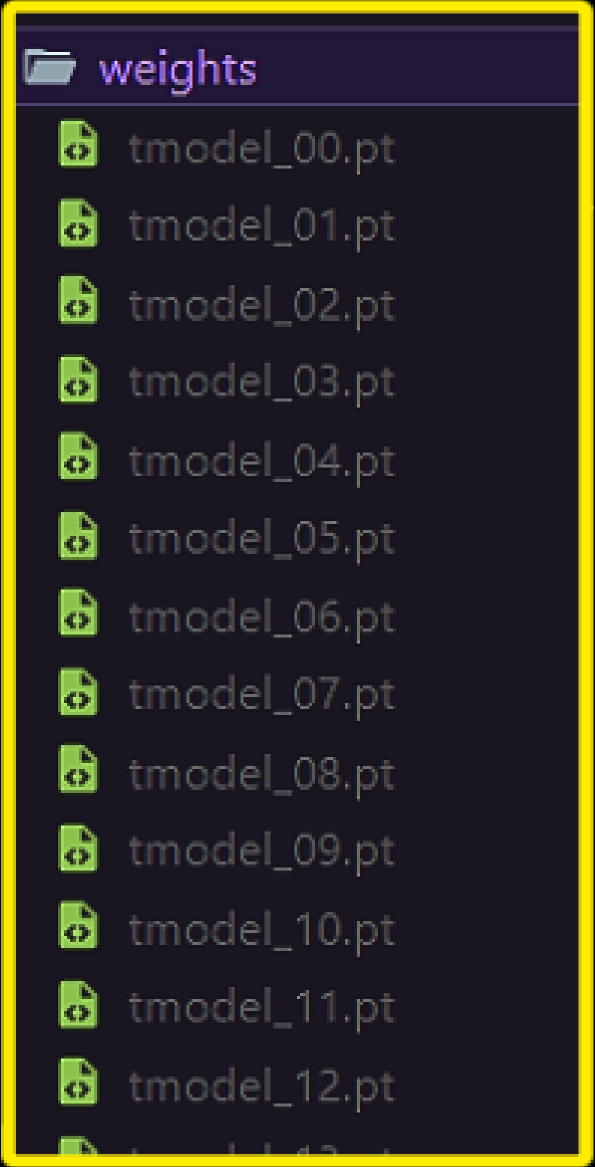


Run ↑	Smoothed	Value	Step	Relative
● runs\tmodel	1.4393	1.4357	67,499	7.384 hr

Without Modular Coding

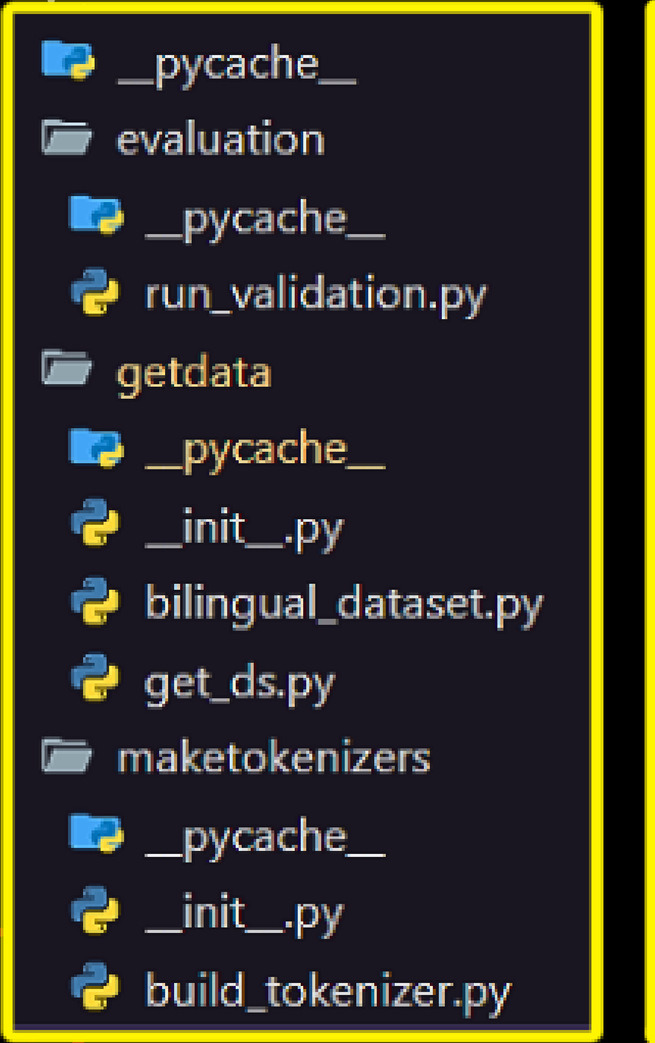


```
Language_Translation_Transformer_Without_Modular_Coding
  runs \ tmodel
    events.out.tfevents.1719074462.DESKTOP-GG8DLS8.10260.0
    events.out.tfevents.1719085322.DESKTOP-GG8DLS8.4188.0
  weights
  tokenizer_en.json
  tokenizer_it.json
  transformer-from-scratch-with-pytorch.ipynb
```

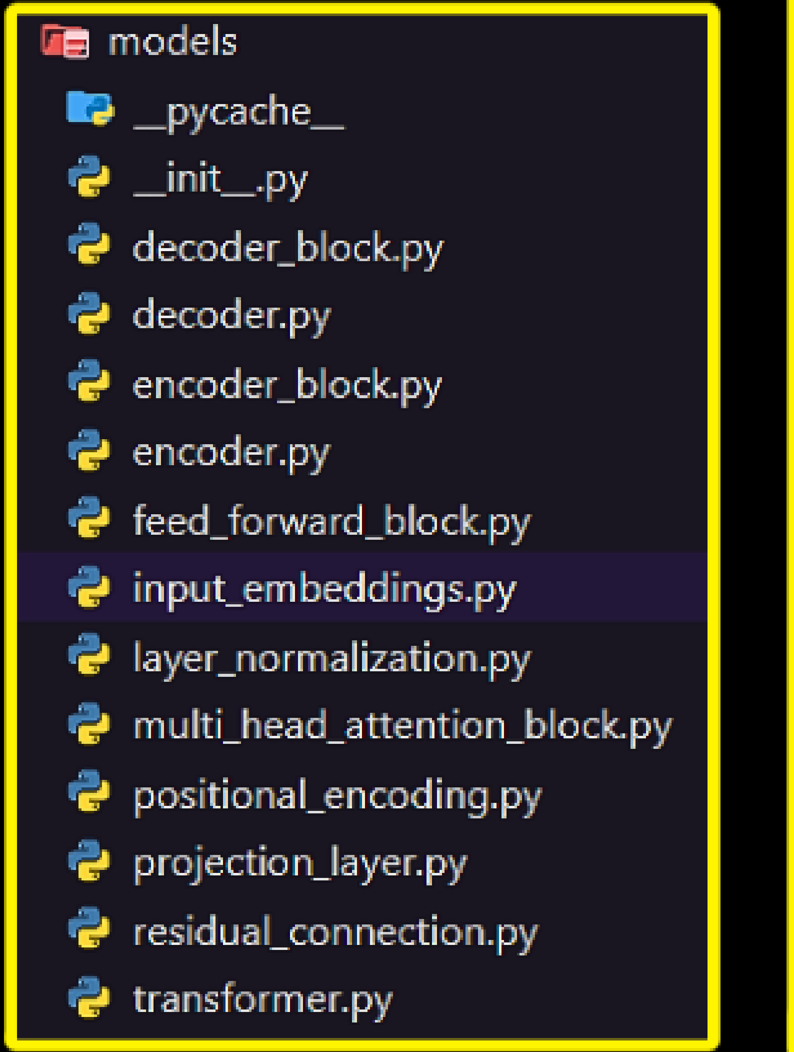


```
weights
  tmodel_00.pt
  tmodel_01.pt
  tmodel_02.pt
  tmodel_03.pt
  tmodel_04.pt
  tmodel_05.pt
  tmodel_06.pt
  tmodel_07.pt
  tmodel_08.pt
  tmodel_09.pt
  tmodel_10.pt
  tmodel_11.pt
  tmodel_12.pt
```

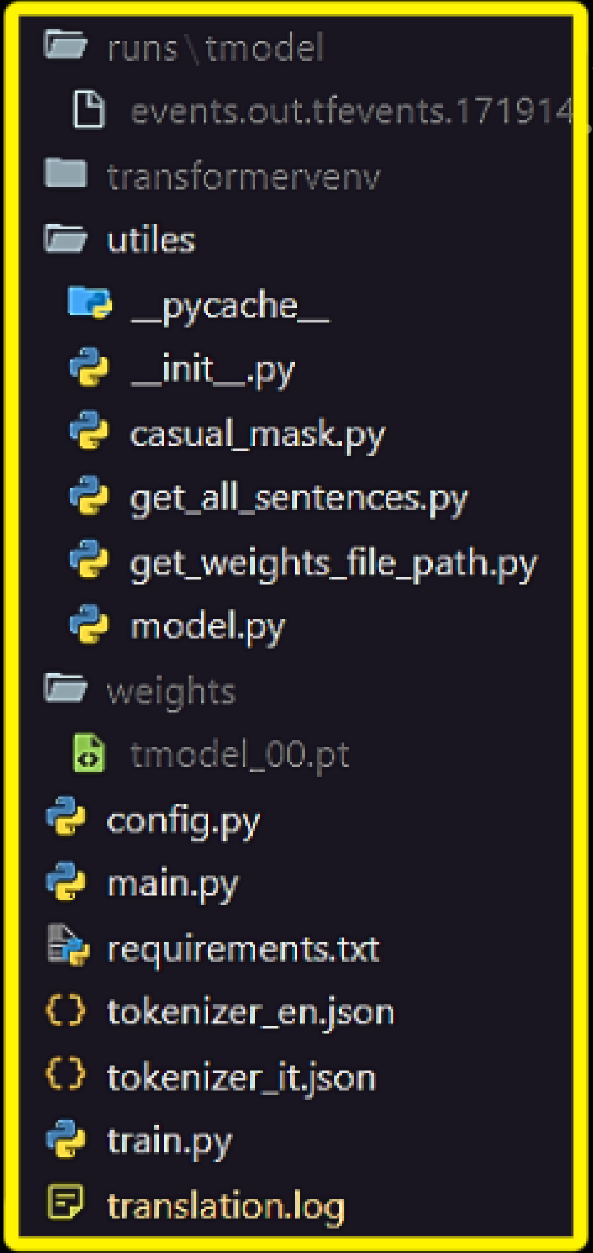
With Modular Coding



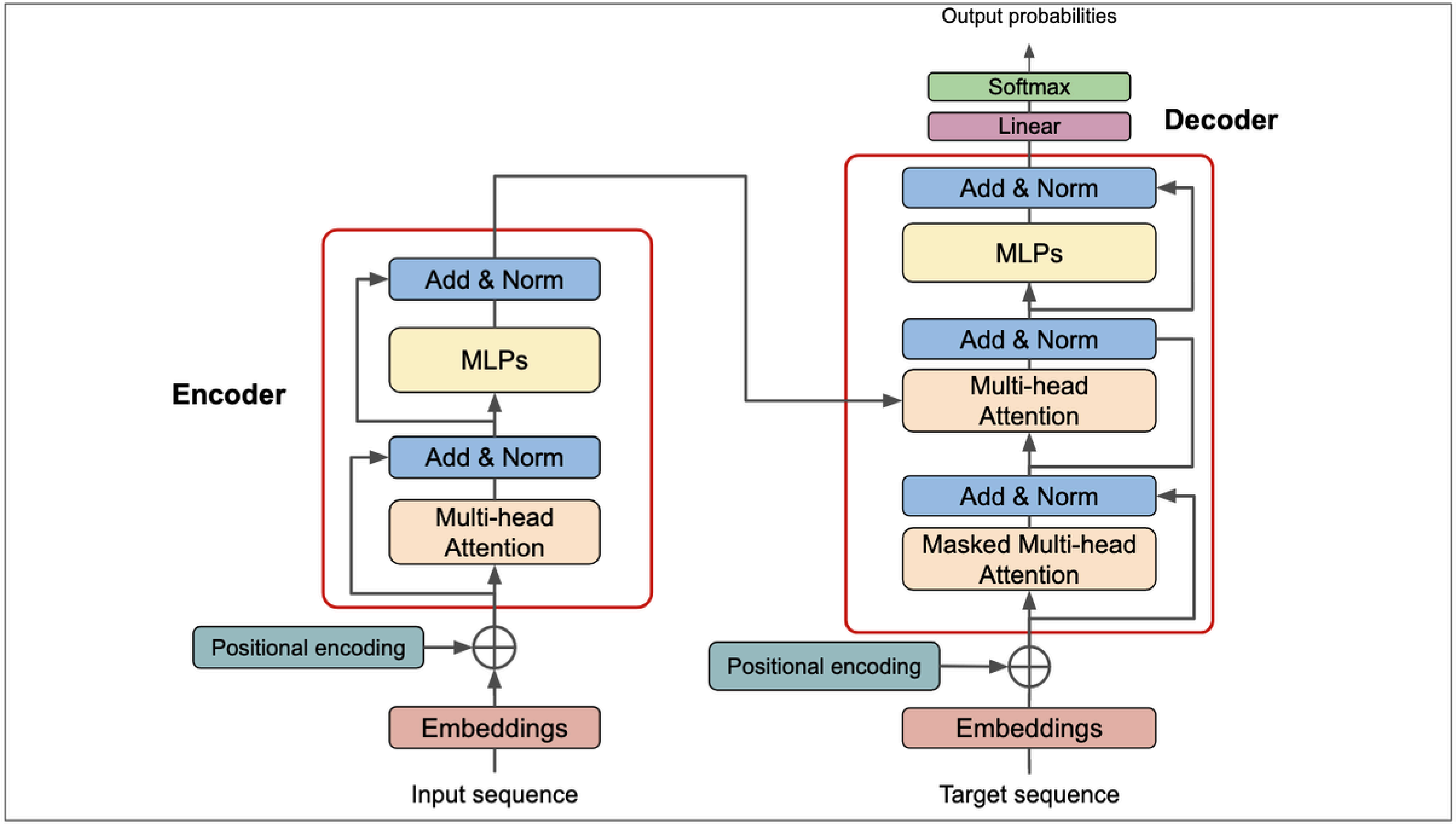
```
__pycache__
evaluation
    __pycache__
    run_validation.py
getdata
    __pycache__
    __init__.py
    bilingual_dataset.py
    get_ds.py
maketokenizers
    __pycache__
    __init__.py
    build_tokenizer.py
```



```
models
    __pycache__
    __init__.py
    decoder_block.py
    decoder.py
    encoder_block.py
    encoder.py
    feed_forward_block.py
    input_embeddings.py
    layer_normalization.py
    multi_head_attention_block.py
    positional_encoding.py
    projection_layer.py
    residual_connection.py
    transformer.py
```



```
runs \ tmodel
    events.out.tfevents.171914
transformervenv
utiles
    __pycache__
    __init__.py
    casual_mask.py
    get_all_sentences.py
    get_weights_file_path.py
    model.py
weights
    tmodel_00.pt
config.py
main.py
requirements.txt
tokenizer_en.json
tokenizer_it.json
train.py
translation.log
```


Output probabilities
Softmax
Linear
Decoder
Add & Norm
MLPs
Add & Norm
Multi-head Attention
Add & Norm
Masked Multi-head Attention
Positional encoding
Embeddings
Target sequence
Encoder
Add & Norm
MLPs
Add & Norm
Multi-head Attention
Positional encoding
Embeddings
Input sequence

Explanation of Code

- Input Embedding: Convert text to number + Normalization
- Position Encoding: Learn Sequence by Adding Zeros
- Layer Normalization: For stability, Convergence speed
- Feed-Forward Network: Use 2 linear functions with a relu activation function, To Introduce Non-Linearity
- Multi-Head Attention: Self-attention (Q, K, V), Contextual Embedding Generated
- Residual Network / Connection: Add/norms, Skip Connection, shortcut for the gradient

- Encoder Block: Feed-Forward Network + Multi-Head Attention + Rresidual Network / Connection
- Encoder: Layer Normalization
- Decoder Block: Feed-Forward Network + Multi-Head Attention + Rresidual Network / Connection + Cross Attention
- Decoder: Layer Normalization
- Project Layer: Converting the output of the model into a probability distribution Using Softmax
- Transformer: Combine all together
- Build Transformation: Define parameters for full operation

- Tokenizer: Word-level tokenization, Convert row text to Number, Mapping
- Get_all_sentences: Extract dataset & Translation
- Get_ds: Getdata, Splitting, Tokenizer
- Create Mask: Creating a Square matrix of size with ones for attention mechanism
- BilingualDataset: Preprocess with special token
- Greedy Decode: Generate the next probable token
- Run Validation: Compare decoder output to the original
- Get mode & Config: Load Model with Config file and define File path in Get_weights_file_path

- Train: Combine all code
- Main: For Training and modular coding
- Weights folder: Model Weights
- Transformerenv: Virtual Environment for this
- Runs folder: Experiment with Tensorboard
- Translation.log: For Logging
- Requirements.txt: Libraries