**Latest Code document**

**tokenizer\_config.yaml**

• Centralized YAML configuration for both text and image processors

• Defines model parameters, preprocessing settings, and special tokens

• Supports BERT tokenizers and BLIP2/CLIP image processors

**text\_tokenizer\_class.py** : Handles text tokenization using HuggingFace transformers. • Loads HuggingFace tokenizers ( BERT)

• text preprocessing (cleaning, normalization, special character handling)

• Configurable tokenization with padding, truncation, tensor conversion

• Returns tokens, token IDs, and processed text

**text\_embedding\_class.py** : Creates embeddings from tokenized text using BERT architecture.

• Custom BERT embedding implementation (BertEmbeddings module)

• Integrates with tokenizer for end-to-end text->embedding conversion

• Handles word embeddings + positional embeddings + layer normalization

• Returns embedding tensors with metadata

**• Flow: Raw text -> Tokenization -> Embedding vectors**

**image\_embeddings.py** : Handles image preprocessing and patch-based embeddings

• ImageProcessor class: BLIP2-based image processing

• PatchEmbed module: Converts images to patch embeddings for Vision Transformer

• Blip2ImageEvalProcessor: Handles image preprocessing (resize, normalize, tensor conversion)

• **Flow: Raw image -> BLIP2 preprocessing -> patch embeddings -> Feature vectors**

**Text Processing Flow:**

1 Tokenizer Class loads BERT tokenizer from config

2 Tokenization: Text -> tokens -> token IDs with padding/truncation

3 Embedding Manager creates BERT embeddings layer

4 Embedding Creation: Token IDs -> contextualized embeddings (768-dim vectors)

**Tokenizer class methods**

**\_load\_config()** : loads the configs from yAML file

**get\_tokenizer\_info()** : information of tokenization from yaml file

**process\_input\_text()**:take input text and process for the following

Strip spaces, Normalize, Lowercase, Remove chars

**load\_tokenizer()** : Get config, Load HF model and Add tokens

**tokenize()** : input processing decision

if yes text processing

apply HF tokenization with apply padding, truncation and return tensor

**EmbeddingManager** **class methods**

**detect\_embedding-type**  : detect the embedding type based on the tokenizer selection

**create\_embedding\_model**  : similarly create model for same

**create\_embeddings** :

* Tokenization via Manager
* Process text
* Get input\_ids from tokenization
* Add metadata to get the embeddings

**load\_embedding\_model()** :

* Load tokenizer,
* Get model config,
* Create BERTmodel,
* Set device to save embeddings

**EmbeddingManager Class flow**

User input(raw text)

|

EmbeddingManager Constructor – (Initialize tokenizer, set variables and load configs)

|

load\_embedding\_model() - (load tokenizer, get model config, create BERT embedding)

|

create\_embeddings() - tokenization via Manager-(process text, get input\_ids)

|

BERT Embeddings (word embed, position embed, layrnorm and drop out)

|

text embedding

**Image Processing Flow:**

1 ImageProcessor loads BLIP2 configuration

2 Preprocessing: Image -> resize (224x224) -> normalize -> tensor

3 Patch Embedding: Image tensor -> 16x16 patches -> 768-dim embeddings per patch

4 Output: Processed image tensor + patch embeddings

**Image\_Processor class methods**

**load\_config():** loads the configs of the processor and patch embeddings

**load\_processor():**

* Get BLIP2 configs
* process the image to resize, normalize and transform to ToTensor

**load\_patch\_embed():** load the configs related to creating patch embeddings

**process\_image()**

* Load image
* convert to RGB and validate the image
* process and create patch embeddings of the image

**Image\_Processor class flow**

User input(image Path)

|

ImageProcessor Constructor - (Load config, initialize processor & patches)

|

load\_processor() - (get the BLIP2 config, preprocess and patches)

|

process\_image() - image loading(PIL.image, convert RGB and validate)

|

BLIP2 transform (resize, ToTensor and Normalize)

|

decision of patch creation

|

patch embedding

**Demo Application (demo.py)**

Main execution flow:

1 User input: Text input

2 Component initialization: Tokenizer, EmbeddingManager, ImageProcessor

3 Model loading: BERT tokenizer/embeddings, BLIP2 processor

4 Text processing: Tokenization -> Embedding generation

5 Image processing: Image loading -> Preprocessing -> patch embeddings