```
import os
os.chdir("florence_model_comparison_visualisation")
os.listdir()
```

```
['trash_percentage_50_text_encoder_florence2_base_attention_map_tensor
.pt',
 'trash_percentage_50_vision_florence2_base_attention_map_tensor.pt',
 'trash_percentage_50_vision_florence2_large_attention_map_tensor.pt',

'fan_percentage_50_text_encoder_florence2_base_attention_map_tensor.pt
',
 'fan_percentage_50_vision_florence2_large_attention_map_tensor.pt',
 'comparison.ipynb',

'trash_percentage_50_text_encoder_florence2_large_attention_map_tensor
.pt',

'fan_percentage_50_text_decoder_florence2_base_attention_map_tensor.pt
',
 'fan_percentage_50_vision_florence2_base_attention_map_tensor.pt',

'trash_percentage_50_text_decoder_florence2_large_attention_map_tensor
.pt',

'fan_percentage_50_text_encoder_florence2_large_attention_map_tensor.p
t',

'fan_percentage_50_text_decoder_florence2_large_attention_map_tensor.p
t',

'trash_percentage_50_text_decoder_florence2_base_attention_map_tensor.
pt']
```

```
import torch

base_model_tensors = {
    'fan':{
        'vision':
torch.load('fan_percentage_50_vision_florence2_base_attention_map_tens
or.pt'),
        'self-attn':
torch.load('fan_percentage_50_text_encoder_florence2_base_attention_ma
p_tensor.pt'),
        'cross-attn':
torch.load('fan_percentage_50_text_decoder_florence2_base_attention_ma
p_tensor.pt')
    },
    'trash':{
        'vision':
```

```
torch.load('trash_percentage_50_vision_florence2_base_attention_map_te
nsor.pt'),
        'self-attn':
torch.load('trash_percentage_50_text_encoder_florence2_base_attention_
map_tensor.pt'),
        'cross-attn':
torch.load('trash_percentage_50_text_decoder_florence2_base_attention_
map_tensor.pt')
    }
}
```

```
/tmp/ipykernel_7899/2802633308.py:5: FutureWarning: You are using
`torch.load` with `weights_only=False` (the current default value),
which uses the default pickle module implicitly. It is possible to
construct malicious pickle data which will execute arbitrary code
during unpickling (See
https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-
models for more details). In a future release, the default value for
`weights_only` will be flipped to `True`. This limits the functions
that could be executed during unpickling. Arbitrary objects will no
longer be allowed to be loaded via this mode unless they are
explicitly allowlisted by the user via
`torch.serialization.add_safe_globals`. We recommend you start setting
`weights_only=True` for any use case where you don't have full control
of the loaded file. Please open an issue on GitHub for any issues
related to this experimental feature.
  'vision':
torch.load('fan_percentage_50_vision_florence2_base_attention_map_tens
or.pt'),
/tmp/ipykernel_7899/2802633308.py:6: FutureWarning: You are using
`torch.load` with `weights_only=False` (the current default value),
which uses the default pickle module implicitly. It is possible to
construct malicious pickle data which will execute arbitrary code
during unpickling (See
https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-
models for more details). In a future release, the default value for
`weights_only` will be flipped to `True`. This limits the functions
that could be executed during unpickling. Arbitrary objects will no
longer be allowed to be loaded via this mode unless they are
explicitly allowlisted by the user via
`torch.serialization.add_safe_globals`. We recommend you start setting
`weights_only=True` for any use case where you don't have full control
of the loaded file. Please open an issue on GitHub for any issues
related to this experimental feature.
  'self-attn':
torch.load('fan_percentage_50_text_encoder_florence2_base_attention_ma
p_tensor.pt'),
/tmp/ipykernel_7899/2802633308.py:7: FutureWarning: You are using
`torch.load` with `weights_only=False` (the current default value),
which uses the default pickle module implicitly. It is possible to
```

```
construct malicious pickle data which will execute arbitrary code
during unpickling (See
https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-
models for more details). In a future release, the default value for
`weights_only` will be flipped to `True`. This limits the functions
that could be executed during unpickling. Arbitrary objects will no
longer be allowed to be loaded via this mode unless they are
explicitly allowlisted by the user via
`torch.serialization.add_safe_globals`. We recommend you start setting
`weights_only=True` for any use case where you don't have full control
of the loaded file. Please open an issue on GitHub for any issues
related to this experimental feature.
  'cross-attn':
torch.load('fan_percentage_50_text_decoder_florence2_base_attention_ma
p_tensor.pt')
/tmp/ipykernel_7899/2802633308.py:10: FutureWarning: You are using
`torch.load` with `weights_only=False` (the current default value),
which uses the default pickle module implicitly. It is possible to
construct malicious pickle data which will execute arbitrary code
during unpickling (See
https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-
models for more details). In a future release, the default value for
`weights_only` will be flipped to `True`. This limits the functions
that could be executed during unpickling. Arbitrary objects will no
longer be allowed to be loaded via this mode unless they are
explicitly allowlisted by the user via
`torch.serialization.add_safe_globals`. We recommend you start setting
`weights_only=True` for any use case where you don't have full control
of the loaded file. Please open an issue on GitHub for any issues
related to this experimental feature.
  'vision':
torch.load('trash_percentage_50_vision_florence2_base_attention_map_te
nsor.pt'),
/tmp/ipykernel_7899/2802633308.py:11: FutureWarning: You are using
`torch.load` with `weights_only=False` (the current default value),
which uses the default pickle module implicitly. It is possible to
construct malicious pickle data which will execute arbitrary code
during unpickling (See
https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-
models for more details). In a future release, the default value for
`weights_only` will be flipped to `True`. This limits the functions
that could be executed during unpickling. Arbitrary objects will no
longer be allowed to be loaded via this mode unless they are
explicitly allowlisted by the user via
`torch.serialization.add_safe_globals`. We recommend you start setting
`weights_only=True` for any use case where you don't have full control
of the loaded file. Please open an issue on GitHub for any issues
related to this experimental feature.
  'self-attn':
```

```
torch.load('trash_percentage_50_text_encoder_florence2_base_attention_
map_tensor.pt'),
/tmp/ipykernel_7899/2802633308.py:12: FutureWarning: You are using
`torch.load` with `weights_only=False` (the current default value),
which uses the default pickle module implicitly. It is possible to
construct malicious pickle data which will execute arbitrary code
during unpickling (See
https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-
models for more details). In a future release, the default value for
`weights_only` will be flipped to `True`. This limits the functions
that could be executed during unpickling. Arbitrary objects will no
longer be allowed to be loaded via this mode unless they are
explicitly allowlisted by the user via
`torch.serialization.add_safe_globals`. We recommend you start setting
`weights_only=True` for any use case where you don't have full control
of the loaded file. Please open an issue on GitHub for any issues
related to this experimental feature.
  'cross-attn':
torch.load('trash_percentage_50_text_decoder_florence2_base_attention_
map_tensor.pt')
```

```
large_model_tensors = {
    'fan':{
        'vision':
torch.load('fan_percentage_50_vision_florence2_large_attention_map_ten
sor.pt'),
        'self-attn':
torch.load('fan_percentage_50_text_encoder_florence2_large_attention_m
ap_tensor.pt'),
        'cross-attn':
torch.load('fan_percentage_50_text_decoder_florence2_large_attention_m
ap_tensor.pt')
    },
    'trash':{
        'vision':
torch.load('trash_percentage_50_vision_florence2_large_attention_map_t
ensor.pt'),
        'self-attn':
torch.load('trash_percentage_50_text_encoder_florence2_large_attention
_map_tensor.pt'),
        'cross-attn':
torch.load('trash_percentage_50_text_decoder_florence2_large_attention
_map_tensor.pt')
    }
}
```

```
/tmp/ipykernel_7899/1391189579.py:3: FutureWarning: You are using
`torch.load` with `weights_only=False` (the current default value),
which uses the default pickle module implicitly. It is possible to
construct malicious pickle data which will execute arbitrary code
```

```
during unpickling (See
https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-
models for more details). In a future release, the default value for
`weights_only` will be flipped to `True`. This limits the functions
that could be executed during unpickling. Arbitrary objects will no
longer be allowed to be loaded via this mode unless they are
explicitly allowlisted by the user via
`torch.serialization.add_safe_globals`. We recommend you start setting
`weights_only=True` for any use case where you don't have full control
of the loaded file. Please open an issue on GitHub for any issues
related to this experimental feature.
  'vision':
torch.load('fan_percentage_50_vision_florence2_large_attention_map_ten
sor.pt'),
/tmp/ipykernel_7899/1391189579.py:4: FutureWarning: You are using
`torch.load` with `weights_only=False` (the current default value),
which uses the default pickle module implicitly. It is possible to
construct malicious pickle data which will execute arbitrary code
during unpickling (See
https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-
models for more details). In a future release, the default value for
`weights_only` will be flipped to `True`. This limits the functions
that could be executed during unpickling. Arbitrary objects will no
longer be allowed to be loaded via this mode unless they are
explicitly allowlisted by the user via
`torch.serialization.add_safe_globals`. We recommend you start setting
`weights_only=True` for any use case where you don't have full control
of the loaded file. Please open an issue on GitHub for any issues
related to this experimental feature.
  'self-attn':
torch.load('fan_percentage_50_text_encoder_florence2_large_attention_m
ap_tensor.pt'),
/tmp/ipykernel_7899/1391189579.py:5: FutureWarning: You are using
`torch.load` with `weights_only=False` (the current default value),
which uses the default pickle module implicitly. It is possible to
construct malicious pickle data which will execute arbitrary code
during unpickling (See
https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-
models for more details). In a future release, the default value for
`weights_only` will be flipped to `True`. This limits the functions
that could be executed during unpickling. Arbitrary objects will no
longer be allowed to be loaded via this mode unless they are
explicitly allowlisted by the user via
`torch.serialization.add_safe_globals`. We recommend you start setting
`weights_only=True` for any use case where you don't have full control
of the loaded file. Please open an issue on GitHub for any issues
related to this experimental feature.
  'cross-attn':
torch.load('fan_percentage_50_text_decoder_florence2_large_attention_m
ap_tensor.pt')
/tmp/ipykernel_7899/1391189579.py:8: FutureWarning: You are using
`torch.load` with `weights_only=False` (the current default value),
which uses the default pickle module implicitly. It is possible to
construct malicious pickle data which will execute arbitrary code
during unpickling (See
https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-
models for more details). In a future release, the default value for
`weights_only` will be flipped to `True`. This limits the functions
that could be executed during unpickling. Arbitrary objects will no
longer be allowed to be loaded via this mode unless they are
explicitly allowlisted by the user via
`torch.serialization.add_safe_globals`. We recommend you start setting
`weights_only=True` for any use case where you don't have full control
of the loaded file. Please open an issue on GitHub for any issues
related to this experimental feature.
  'vision':
torch.load('trash_percentage_50_vision_florence2_large_attention_map_t
ensor.pt'),
/tmp/ipykernel_7899/1391189579.py:9: FutureWarning: You are using
`torch.load` with `weights_only=False` (the current default value),
which uses the default pickle module implicitly. It is possible to
construct malicious pickle data which will execute arbitrary code
during unpickling (See
https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-
models for more details). In a future release, the default value for
`weights_only` will be flipped to `True`. This limits the functions
that could be executed during unpickling. Arbitrary objects will no
longer be allowed to be loaded via this mode unless they are
explicitly allowlisted by the user via
`torch.serialization.add_safe_globals`. We recommend you start setting
`weights_only=True` for any use case where you don't have full control
of the loaded file. Please open an issue on GitHub for any issues
related to this experimental feature.
  'self-attn':
torch.load('trash_percentage_50_text_encoder_florence2_large_attention
_map_tensor.pt'),
/tmp/ipykernel_7899/1391189579.py:10: FutureWarning: You are using
`torch.load` with `weights_only=False` (the current default value),
which uses the default pickle module implicitly. It is possible to
construct malicious pickle data which will execute arbitrary code
during unpickling (See
https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-
models for more details). In a future release, the default value for
`weights_only` will be flipped to `True`. This limits the functions
that could be executed during unpickling. Arbitrary objects will no
longer be allowed to be loaded via this mode unless they are
explicitly allowlisted by the user via
`torch.serialization.add_safe_globals`. We recommend you start setting
```

```
`weights_only=True` for any use case where you don't have full control
of the loaded file. Please open an issue on GitHub for any issues
related to this experimental feature.
  'cross-attn':
torch.load('trash_percentage_50_text_decoder_florence2_large_attention
_map_tensor.pt')
```

```
base_model_tensors['fan']['vision'].shape, base_model_tensors['fan']
['self-attn'].shape, base_model_tensors['fan']['cross-attn'].shape
```

```
(torch.Size([1, 9216, 256]),
 torch.Size([1, 584, 768]),
 torch.Size([3, 1, 768]))
```

```
large_model_tensors['fan']['vision'].shape, large_model_tensors['fan']
['self-attn'].shape, large_model_tensors['fan']['cross-attn'].shape
```

```
(torch.Size([1, 9216, 512]),
 torch.Size([1, 584, 1024]),
 torch.Size([3, 1, 1024]))
```

```
!pip install umap-learn
```

```
Collecting umap-learn
  Downloading umap_learn-0.5.7-py3-none-any.whl.metadata (21 kB)
Requirement already satisfied: numpy>=1.17 in
/opt/conda/lib/python3.11/site-packages (from umap-learn) (2.1.2)
Requirement already satisfied: scipy>=1.3.1 in
/opt/conda/lib/python3.11/site-packages (from umap-learn) (1.15.1)
Requirement already satisfied: scikit-learn>=0.22 in
/opt/conda/lib/python3.11/site-packages (from umap-learn) (1.6.1)
Collecting numba>=0.51.2 (from umap-learn)
  Downloading numba-0.61.0-cp311-cp311-
manylinux2014_x86_64.manylinux_2_17_x86_64.whl.metadata (2.8 kB)
Collecting pynndescent>=0.5 (from umap-learn)
  Downloading pynndescent-0.5.13-py3-none-any.whl.metadata (6.8 kB)
Requirement already satisfied: tqdm in /opt/conda/lib/python3.11/site-
packages (from umap-learn) (4.66.5)
Collecting llvmlite<0.45,>=0.44.0dev0 (from numba>=0.51.2->umap-learn)
  Downloading llvmlite-0.44.0-cp311-cp311-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (4.8 kB)
Requirement already satisfied: joblib>=0.11 in
/opt/conda/lib/python3.11/site-packages (from pynndescent>=0.5->umap-
learn) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in
/opt/conda/lib/python3.11/site-packages (from scikit-learn>=0.22-
>umap-learn) (3.5.0)
Downloading umap_learn-0.5.7-py3-none-any.whl (88 kB)
Downloading numba-0.61.0-cp311-cp311-
manylinux2014_x86_64.manylinux_2_17_x86_64.whl (3.8 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 3.8/3.8 MB 62.8 MB/s eta
```

```
0:00:00
lite-0.44.0-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl
(42.4 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 42.4/42.4 MB 60.6 MB/s eta
0:00:0000:0100:01
lite, numba, pynndescent, umap-learn
Successfully installed llvmlite-0.44.0 numba-0.61.0 pynndescent-0.5.13
umap-learn-0.5.7
WARNING: Running pip as the 'root' user can result in broken
permissions and conflicting behaviour with the system package manager,
possibly rendering your system unusable.It is recommended to use a
virtual environment instead: https://pip.pypa.io/warnings/venv. Use
the --root-user-action option if you know what you are doing and want
to suppress this warning.

[notice] A new release of pip is available: 24.3.1 -> 25.0
[notice] To update, run: pip install --upgrade pip
```

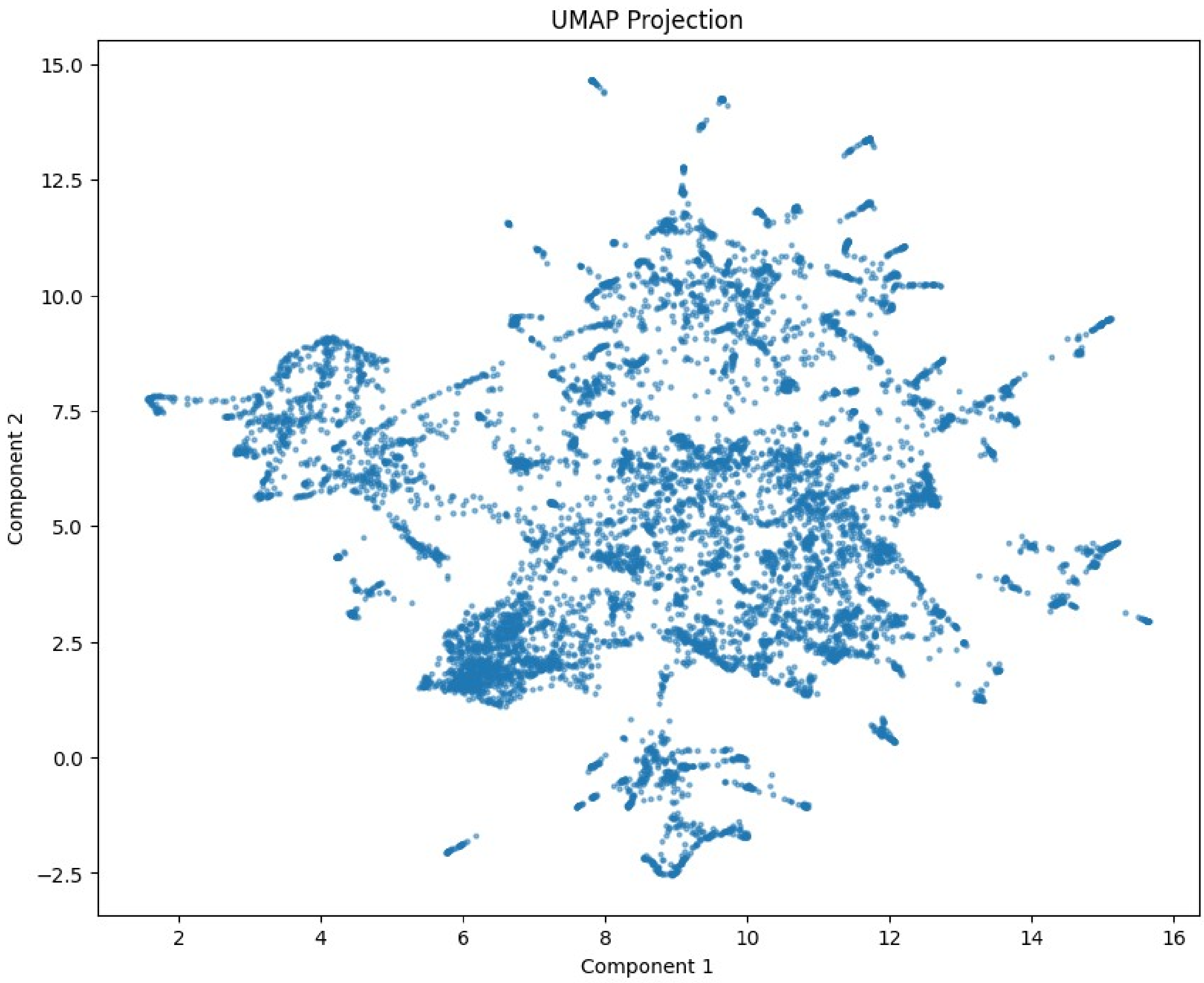
```python
import torch

# Assuming your tensor is named 'data_tensor'
data = base_model_tensors['fan']['vision'].squeeze(0)  # Removes the first dimension [1, 9216, 256] → [9216, 256]

# If the tensor is on GPU, move to CPU and convert to NumPy
data_np = data.cpu().detach().numpy()

import umap.umap_ as umap

# Initialize UMAP reducer (customize parameters if needed)
reducer = umap.UMAP(n_neighbors=15, min_dist=0.1, n_components=2)

# Fit UMAP to the data and transform to 2D
embedding = reducer.fit_transform(data_np)
```

```
/opt/conda/lib/python3.11/site-packages/sklearn/utils/
deprecation.py:151: FutureWarning: 'force_all_finite' was renamed to
'ensure_all_finite' in 1.6 and will be removed in 1.8.
  warnings.warn(
```

```python
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
data_scaled = scaler.fit_transform(data_np)
embedding = reducer.fit_transform(data_scaled)  # Use scaled data
```

```
/opt/conda/lib/python3.11/site-packages/sklearn/utils/
deprecation.py:151: FutureWarning: 'force_all_finite' was renamed to
'ensure_all_finite' in 1.6 and will be removed in 1.8.
  warnings.warn(
```

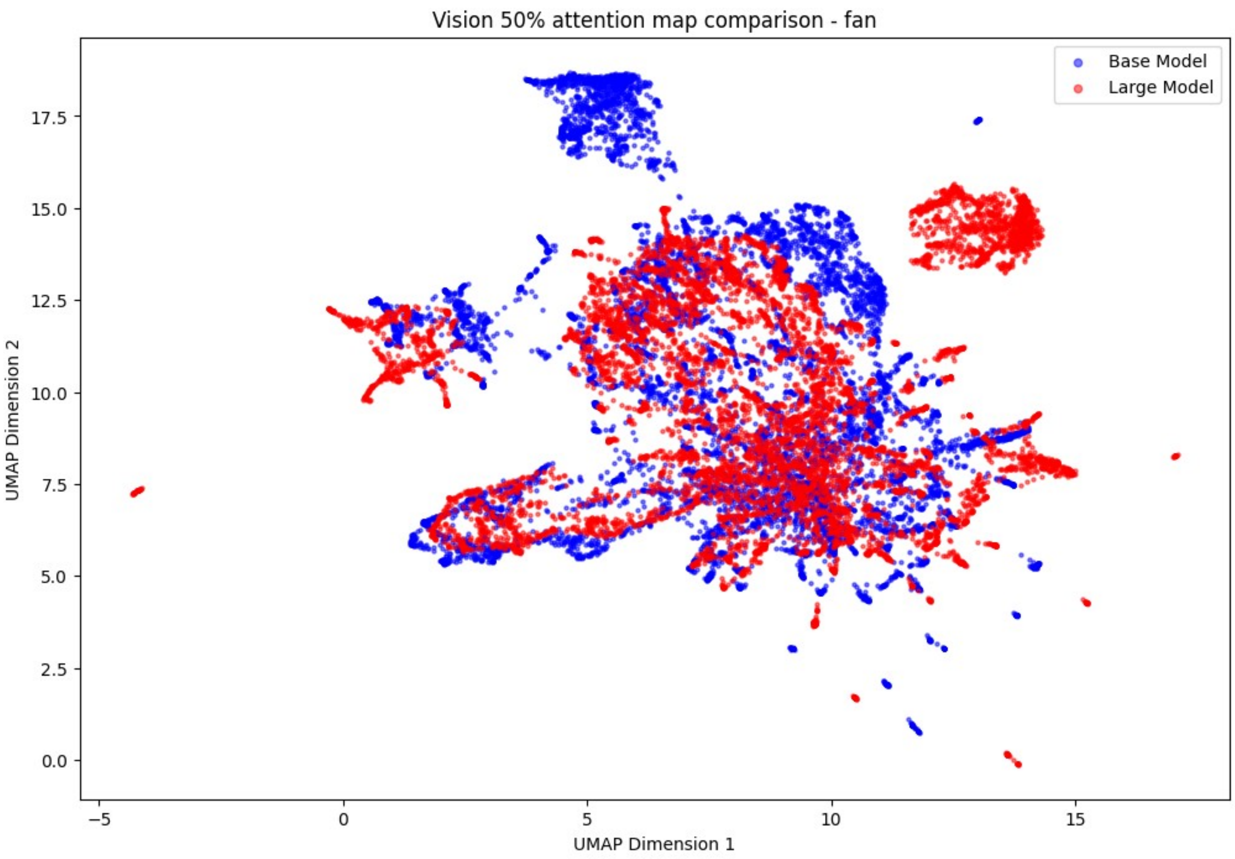
```python
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 8))
plt.scatter(embedding[:, 0], embedding[:, 1], s=5, alpha=0.5)
plt.title("UMAP Projection")
plt.xlabel("Component 1")
plt.ylabel("Component 2")
plt.show()
```



```python
import torch
import umap.umap_ as umap

def reduce_to_2d_umap(data_tensor, **umap_kwargs):
    """
    Converts a PyTorch tensor to a 2D UMAP embedding.
    Args:
        data_tensor: Input tensor with shape [batch_size, num_samples,
num_features]
                    (batch_size should be 1)
        **umap_kwargs: Optional UMAP parameters (e.g., n_neighbors,
min_dist)

    Returns:
        numpy.ndarray: 2D embedding with shape [num_samples, 2]
    """
    # Remove batch dimension and convert to NumPy
    data_np = data_tensor.squeeze(0).cpu().detach().numpy()

    # Force 2D output (override any n_components argument)
    umap_kwargs.pop('n_components', None)

    # Create and fit UMAP reducer
    reducer = umap.UMAP(n_components=2, **umap_kwargs)
    embedding = reducer.fit_transform(data_np)

    return embedding

# Assuming your tensor is named `input_tensor` with shape [1, 9216,
256]
embedding_2d_base = reduce_to_2d_umap(
    base_model_tensors['fan']['vision'],
    n_neighbors=20,  # Example custom parameter
    min_dist=0.1     # Another example parameter
)

embedding_2d_large = reduce_to_2d_umap(
    large_model_tensors['fan']['vision'],
    n_neighbors=20,  # Example custom parameter
    min_dist=0.1     # Another example parameter
)

# # To plot (requires matplotlib):
# import matplotlib.pyplot as plt
# plt.scatter(embedding_2d[:, 0], embedding_2d[:, 1], s=2)
# plt.show()
```

```
/opt/conda/lib/python3.11/site-packages/sklearn/utils/
deprecation.py:151: FutureWarning: 'force_all_finite' was renamed to
'ensure_all_finite' in 1.6 and will be removed in 1.8.
  warnings.warn(
/opt/conda/lib/python3.11/site-packages/sklearn/utils/deprecation.py:1
51: FutureWarning: 'force_all_finite' was renamed to
'ensure_all_finite' in 1.6 and will be removed in 1.8.
  warnings.warn(
```

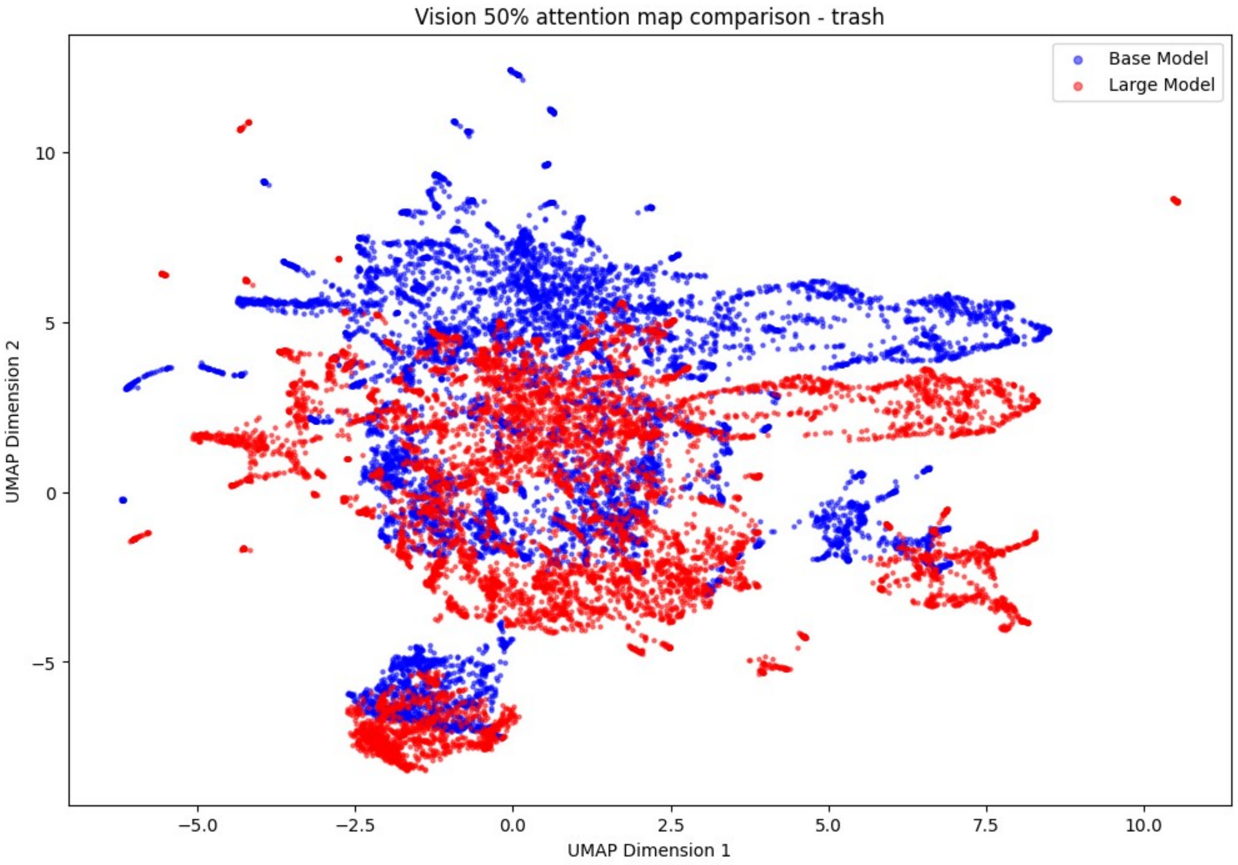
```python
import matplotlib.pyplot as plt

def plot_embeddings_2d(embedding1, embedding2,
                       labels=('Base Model', 'Large Model'),
                       colors=('blue', 'red'),
                       alpha=0.5, s=5, figsize=(12, 8),
                       title='UMAP Projection Comparison'):
    """
    Plots two 2D embeddings in the same scatter plot with different colors.

    Args:
        embedding1: First 2D embedding should be from base model
        embedding2: Second 2D embedding should be from large model
        labels: Tuple of legend labels (default: ('Base Model', 'Large Model'))
        colors: Tuple of colors for each embedding (default: ('blue', 'red'))
        alpha: Transparency of points (0-1, default: 0.5)
        s: Marker size (default: 5)
        figsize: Figure size (default: (12, 8))
        title: Plot title (default: 'UMAP Projection Comparison')
    """
    plt.figure(figsize=figsize)

    # Plot first embedding
    plt.scatter(embedding1[:, 0], embedding1[:, 1],
                c=colors[0], label=labels[0],
                alpha=alpha, s=s)

    # Plot second embedding
    plt.scatter(embedding2[:, 0], embedding2[:, 1],
                c=colors[1], label=labels[1],
                alpha=alpha, s=s)

    plt.title(title)
    plt.xlabel('UMAP Dimension 1')
    plt.ylabel('UMAP Dimension 2')
    plt.legend(markerscale=2)  # Make legend markers larger than points
    plt.show()

plot_embeddings_2d(embedding_2d_base, embedding_2d_large,
title=r'Vision 50% attention map comparison - fan')
```



```
# Assuming your tensor is named `input_tensor` with shape [1, 9216,
256]
embedding_2d_base = reduce_to_2d_umap(
    base_model_tensors['trash']['vision'],
    n_neighbors=20,  # Example custom parameter
    min_dist=0.1     # Another example parameter
)

embedding_2d_large = reduce_to_2d_umap(
    large_model_tensors['trash']['vision'],
    n_neighbors=20,  # Example custom parameter
    min_dist=0.1     # Another example parameter
)

# # To plot (requires matplotlib):
# import matplotlib.pyplot as plt
# plt.scatter(embedding_2d[:, 0], embedding_2d[:, 1], s=2)
# plt.show()

plot_embeddings_2d(embedding_2d_base, embedding_2d_large,
title=r'Vision 50% attention map comparison - trash')
```

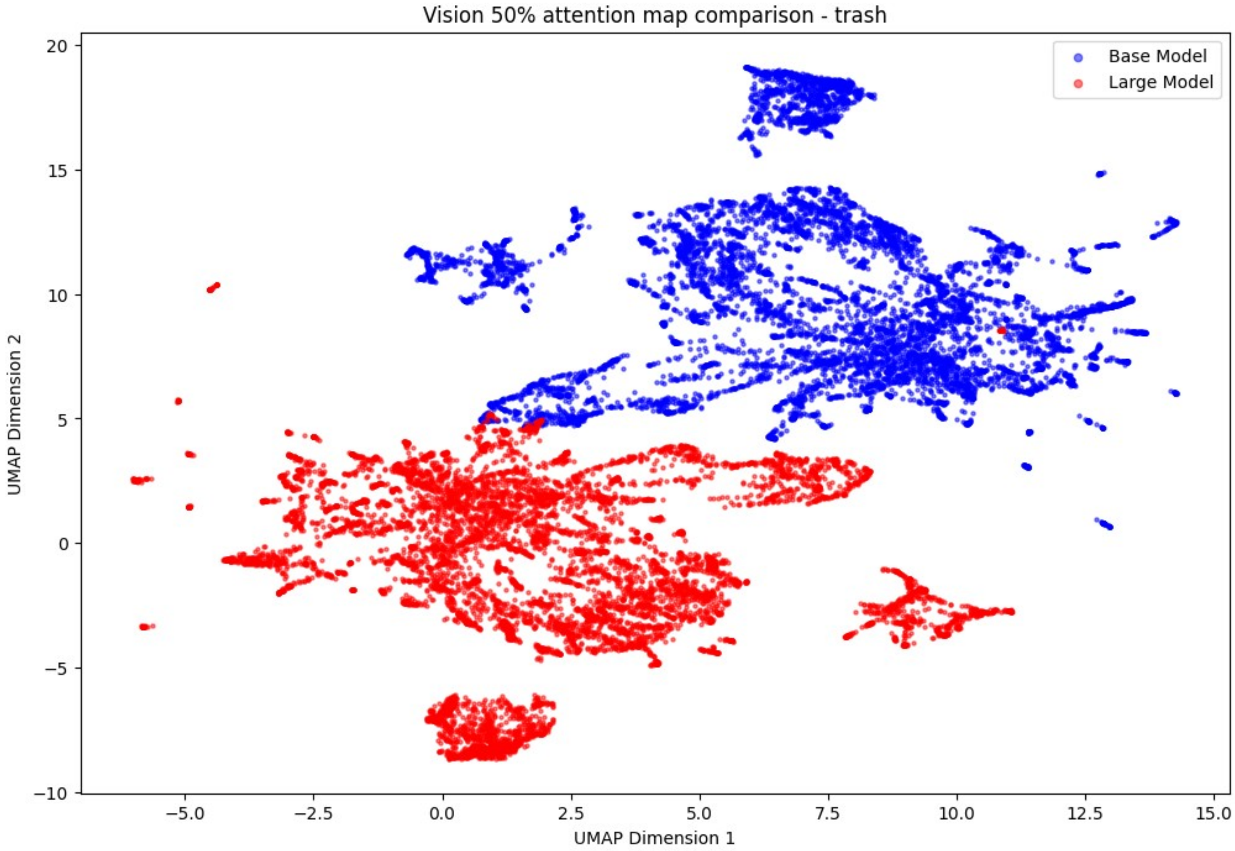
```
/opt/conda/lib/python3.11/site-packages/sklearn/utils/
deprecation.py:151: FutureWarning: 'force_all_finite' was renamed to
```

```
'ensure_all_finite' in 1.6 and will be removed in 1.8.
  warnings.warn(
/opt/conda/lib/python3.11/site-packages/sklearn/utils/deprecation.py:1
51: FutureWarning: 'force_all_finite' was renamed to
'ensure_all_finite' in 1.6 and will be removed in 1.8.
  warnings.warn(
```



```
# Assuming your tensor is named `input_tensor` with shape [1, 9216,
256]
embedding_2d_base = reduce_to_2d_umap(
    base_model_tensors['trash']['vision'],
    n_neighbors=20,  # Example custom parameter
    min_dist=0.1     # Another example parameter
)

embedding_2d_large = reduce_to_2d_umap(
    large_model_tensors['trash']['vision'],
    n_neighbors=20,  # Example custom parameter
    min_dist=0.1     # Another example parameter
)

# # To plot (requires matplotlib):
# import matplotlib.pyplot as plt
# plt.scatter(embedding_2d[:, 0], embedding_2d[:, 1], s=2)
# plt.show()

plot_embeddings_2d(embedding_2d_base, embedding_2d_large,
title=r'Vision 50% attention map comparison - trash')
```

```
/opt/conda/lib/python3.11/site-packages/sklearn/utils/
deprecation.py:151: FutureWarning: 'force_all_finite' was renamed to
'ensure_all_finite' in 1.6 and will be removed in 1.8.
  warnings.warn(
/opt/conda/lib/python3.11/site-packages/sklearn/utils/deprecation.py:1
51: FutureWarning: 'force_all_finite' was renamed to
'ensure_all_finite' in 1.6 and will be removed in 1.8.
  warnings.warn(
```

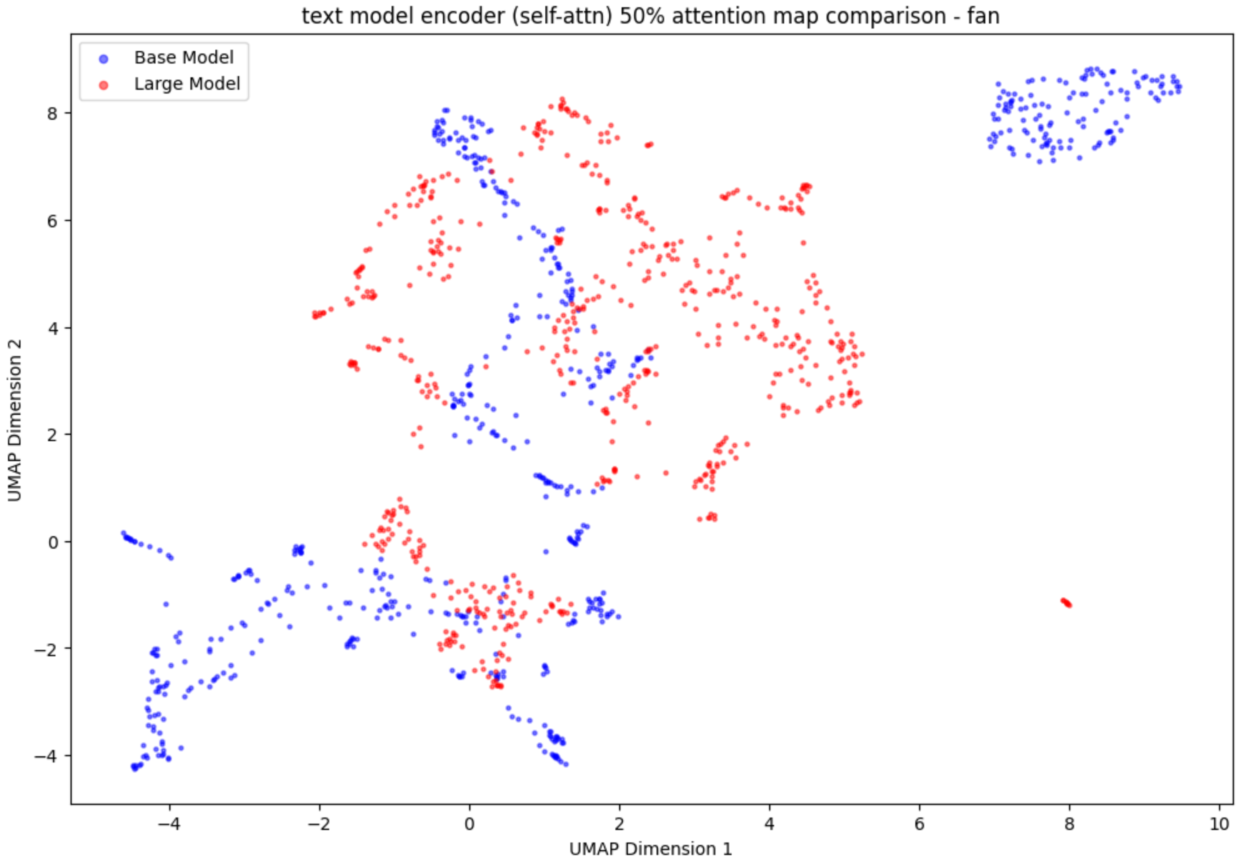


Self-attension maps

```
# Assuming your tensor is named `input_tensor` with shape [1, 9216,
256]
embedding_2d_base = reduce_to_2d_umap(
    base_model_tensors['fan']['self-attn'],
    n_neighbors=20,  # Example custom parameter
    min_dist=0.1     # Another example parameter
```

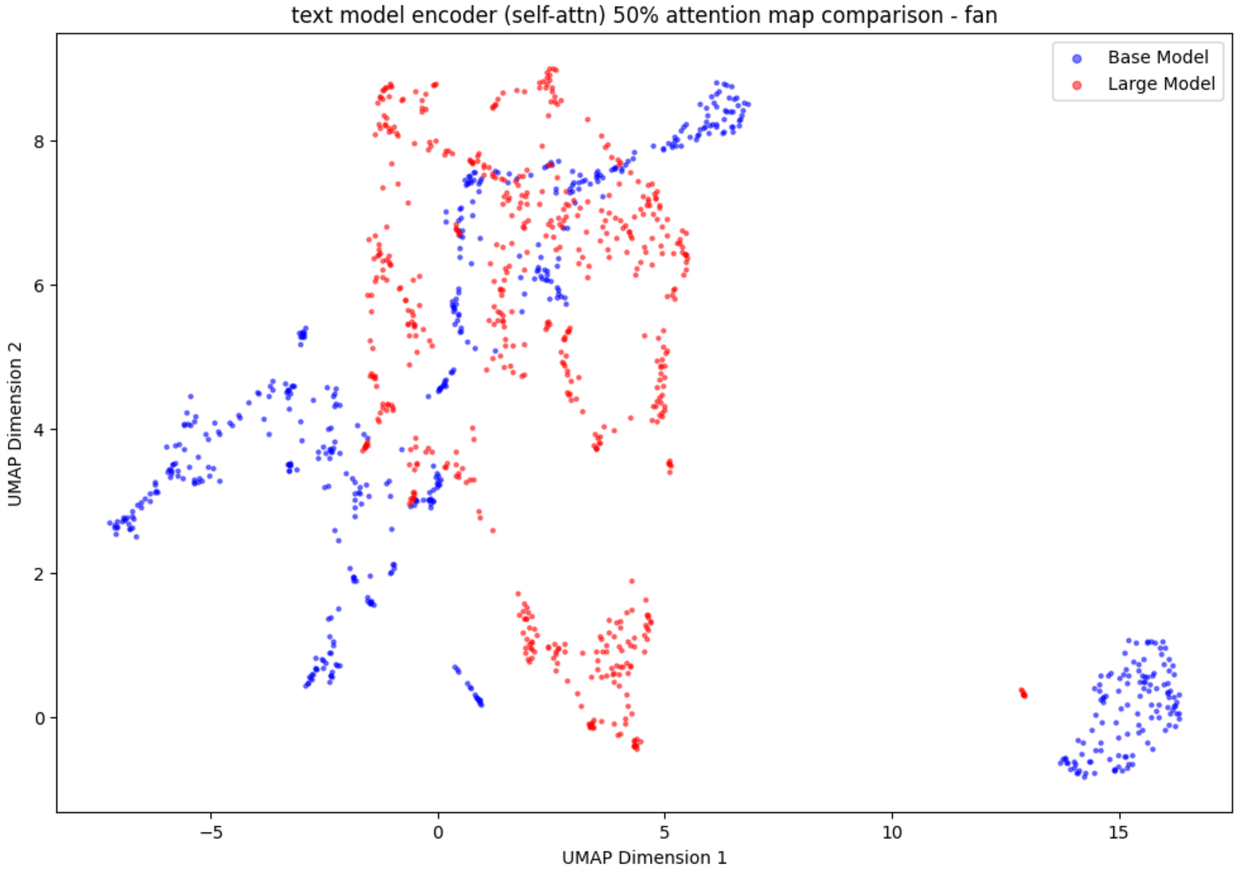
```python
)

embedding_2d_large = reduce_to_2d_umap(
    large_model_tensors['fan']['self-attn'],
    n_neighbors=20,  # Example custom parameter
    min_dist=0.1     # Another example parameter
)

# # To plot (requires matplotlib):
# import matplotlib.pyplot as plt
# plt.scatter(embedding_2d[:, 0], embedding_2d[:, 1], s=2)
# plt.show()

plot_embeddings_2d(embedding_2d_base, embedding_2d_large, title=r'text
model encoder (self-attn) 50% attention map comparison - fan')
```

```
/opt/conda/lib/python3.11/site-packages/sklearn/utils/
deprecation.py:151: FutureWarning: 'force_all_finite' was renamed to
'ensure_all_finite' in 1.6 and will be removed in 1.8.
  warnings.warn(
/opt/conda/lib/python3.11/site-packages/sklearn/utils/deprecation.py:1
51: FutureWarning: 'force_all_finite' was renamed to
'ensure_all_finite' in 1.6 and will be removed in 1.8.
  warnings.warn(
```



```
# Assuming your tensor is named `input_tensor` with shape [1, 9216,
256]
embedding_2d_base = reduce_to_2d_umap(
    base_model_tensors['fan']['self-attn'],
    n_neighbors=20,  # Example custom parameter
    min_dist=0.1     # Another example parameter
)

embedding_2d_large = reduce_to_2d_umap(
    large_model_tensors['fan']['self-attn'],
    n_neighbors=20,  # Example custom parameter
    min_dist=0.1     # Another example parameter
)

# # To plot (requires matplotlib):
# import matplotlib.pyplot as plt
# plt.scatter(embedding_2d[:, 0], embedding_2d[:, 1], s=2)
# plt.show()

plot_embeddings_2d(embedding_2d_base, embedding_2d_large, title=r'text
model encoder (self-attn) 50% attention map comparison - fan')
```

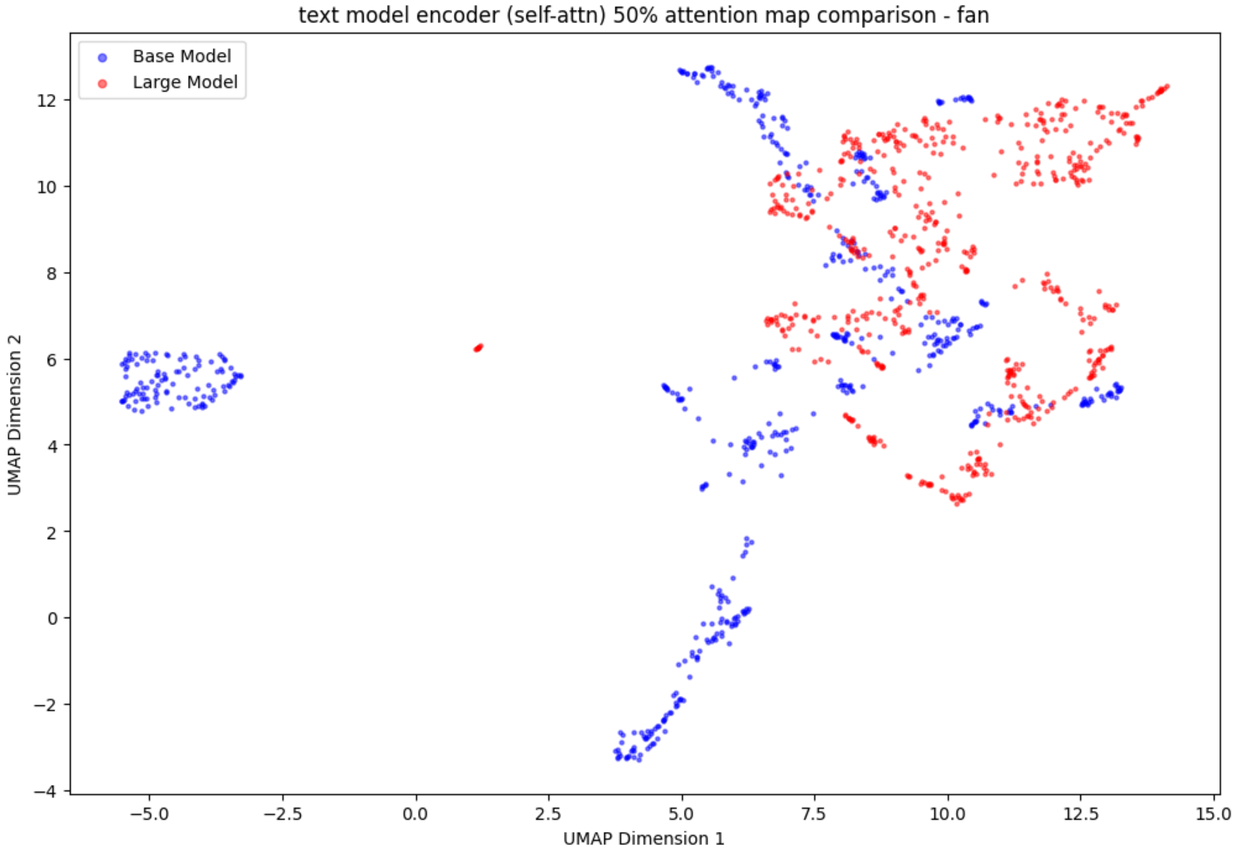
```
/opt/conda/lib/python3.11/site-packages/sklearn/utils/
deprecation.py:151: FutureWarning: 'force_all_finite' was renamed to
'ensure_all_finite' in 1.6 and will be removed in 1.8.
  warnings.warn(
/opt/conda/lib/python3.11/site-packages/sklearn/utils/deprecation.py:1
51: FutureWarning: 'force_all_finite' was renamed to
'ensure_all_finite' in 1.6 and will be removed in 1.8.
  warnings.warn(
```



```python
# Assuming your tensor is named `input_tensor` with shape [1, 9216,
256]
embedding_2d_base = reduce_to_2d_umap(
    base_model_tensors['trash']['self-attn'],
    n_neighbors=20,  # Example custom parameter
    min_dist=0.1     # Another example parameter
)

embedding_2d_large = reduce_to_2d_umap(
    large_model_tensors['trash']['self-attn'],
    n_neighbors=20,  # Example custom parameter
    min_dist=0.1     # Another example parameter
)
```

```
# # To plot (requires matplotlib):
# import matplotlib.pyplot as plt
# plt.scatter(embedding_2d[:, 0], embedding_2d[:, 1], s=2)
# plt.show()

plot_embeddings_2d(embedding_2d_base, embedding_2d_large, title=r'text
model encoder (self-attn) 50% attention map comparison - fan')
```

```
/opt/conda/lib/python3.11/site-packages/sklearn/utils/
deprecation.py:151: FutureWarning: 'force_all_finite' was renamed to
'ensure_all_finite' in 1.6 and will be removed in 1.8.
  warnings.warn(
/opt/conda/lib/python3.11/site-packages/sklearn/utils/deprecation.py:1
51: FutureWarning: 'force_all_finite' was renamed to
'ensure_all_finite' in 1.6 and will be removed in 1.8.
  warnings.warn(
```



Cross Attention

```
base_model_tensors['fan']['cross-attn'].shape
```

```
torch.Size([3, 1, 768])
```

```python
base_model_tensors['fan']['cross-attn'][0].shape
```

```
torch.Size([1, 768])
```

```python
# Assuming your tensor is named `input_tensor` with shape [1, 9216,
256]
embedding_2d_base = reduce_to_2d_umap(
    base_model_tensors['fan']['cross-attn'],
    n_neighbors=20,  # Example custom parameter
    min_dist=0.1     # Another example parameter
)

embedding_2d_large = reduce_to_2d_umap(
    large_model_tensors['fan']['cross-attn'],
    n_neighbors=20,  # Example custom parameter
    min_dist=0.1     # Another example parameter
)

# # To plot (requires matplotlib):
# import matplotlib.pyplot as plt
# plt.scatter(embedding_2d[:, 0], embedding_2d[:, 1], s=2)
# plt.show()

plot_embeddings_2d(embedding_2d_base, embedding_2d_large, title=r'text
model decoder (cross-attn) 50% attention map comparison - fan')
```

```
/opt/conda/lib/python3.11/site-packages/sklearn/utils/
deprecation.py:151: FutureWarning: 'force_all_finite' was renamed to
'ensure_all_finite' in 1.6 and will be removed in 1.8.
  warnings.warn(
```

```
----------------------------------------------------------------------
-----
ValueError                                Traceback (most recent call
last)
Cell In[34], line 2
      1 # Assuming your tensor is named `input_tensor` with shape [1,
9216, 256]
----> 2 embedding_2d_base = reduce_to_2d_umap(
      3     base_model_tensors['fan']['cross-attn'],
      4     n_neighbors=20,  # Example custom parameter
      5     min_dist=0.1     # Another example parameter
      6 )
      8 embedding_2d_large = reduce_to_2d_umap(
      9     large_model_tensors['fan']['cross-attn'],
     10     n_neighbors=20,  # Example custom parameter
     11     min_dist=0.1     # Another example parameter
     12 )
     14 # # To plot (requires matplotlib):
     15 # import matplotlib.pyplot as plt
     16 # plt.scatter(embedding_2d[:, 0], embedding_2d[:, 1], s=2)
     17 # plt.show()

Cell In[15], line 24, in reduce_to_2d_umap(data_tensor, **umap_kwargs)
     22 # Create and fit UMAP reducer
     23 reducer = umap.UMAP(n_components=2, **umap_kwargs)
---> 24 embedding = reducer.fit_transform(data_np)
     26 return embedding

File /opt/conda/lib/python3.11/site-packages/umap/umap_.py:2928, in
UMAP.fit_transform(self, X, y, force_all_finite, **kwargs)
   2890 def fit_transform(self, X, y=None, force_all_finite=True,
**kwargs):
   2891     """Fit X into an embedded space and return that
transformed
   2892     output.
   2893
   (...)
   2926         Local radii of data points in the embedding (log-
transformed).
   2927     """
-> 2928     self.fit(X, y, force_all_finite, **kwargs)
   2929     if self.transform_mode == "embedding":
   2930         if self.output_dens:

File /opt/conda/lib/python3.11/site-packages/umap/umap_.py:2372, in
UMAP.fit(self, X, y, force_all_finite, **kwargs)
   2368     X = check_array(
   2369         X, dtype=np.uint8, order="C",
force_all_finite=force_all_finite
   2370     )
   2371 else:
-> 2372     X = check_array(
   2373         X,
   2374         dtype=np.float32,
   2375         accept_sparse="csr",
   2376         order="C",
   2377         force_all_finite=force_all_finite,
   2378     )
   2379 self._raw_data = X
   2381 # Handle all the optional arguments, setting default

File
/opt/conda/lib/python3.11/site-packages/sklearn/utils/validation.py:11
01, in check_array(array, accept_sparse, accept_large_sparse, dtype,
order, copy, force_writeable, force_all_finite, ensure_all_finite,
ensure_non_negative, ensure_2d, allow_nd, ensure_min_samples,
ensure_min_features, estimator, input_name)
   1096     raise ValueError(
   1097         "dtype='numeric' is not compatible with arrays of
bytes/strings."
   1098         "Convert your data to numeric values explicitly
instead."
   1099     )
   1100 if not allow_nd and array.ndim >= 3:
-> 1101     raise ValueError(
   1102         "Found array with dim %d. %s expected <= 2."
   1103         % (array.ndim, estimator_name)
   1104     )
   1106 if ensure_all_finite:
   1107     _assert_all_finite(
   1108         array,
   1109         input_name=input_name,
   1110         estimator_name=estimator_name,
   1111         allow_nan=ensure_all_finite == "allow-nan",
   1112     )

ValueError: Found array with dim 3. None expected <= 2.

# Assuming your tensor is named `input_tensor` with shape [1, 9216,
256]
embedding_2d_base = reduce_to_2d_umap(
    base_model_tensors['fan']['self-attn'],
    n_neighbors=20,  # Example custom parameter
    min_dist=0.1     # Another example parameter
)

embedding_2d_large = reduce_to_2d_umap(
    large_model_tensors['fan']['self-attn'],
    n_neighbors=20,  # Example custom parameter
    min_dist=0.1     # Another example parameter
)

# # To plot (requires matplotlib):
# import matplotlib.pyplot as plt
# plt.scatter(embedding_2d[:, 0], embedding_2d[:, 1], s=2)
# plt.show()

plot_embeddings_2d(embedding_2d_base, embedding_2d_large, title=r'text
model encoder (self-attn) 50% attention map comparison - fan')
```

```
import torch
import umap
import matplotlib.pyplot as plt

def reduce_dimensions(tensor):
    # Ensure tensor is 2D (samples x features)
```

```python
    if len(tensor.shape) > 2:
        tensor = tensor.squeeze(0)

    # Convert PyTorch tensor to NumPy array
    data_array = tensor.numpy()

    # Apply UMAP
    reducer = umap.UMAP(n_components=2, random_state=42)
    reduced_data = reducer.fit_transform(data_array)

    # Plot the reduced data
    plt.figure(figsize=(10, 8))
    plt.scatter(reduced_data[:, 0], reduced_data[:, 1], alpha=0.7)
    plt.title('UMAP Dimensionality Reduction')
    plt.xlabel('UMAP Dimension 1')
    plt.ylabel('UMAP Dimension 2')
    plt.tight_layout()
    plt.show()

    return reduced_data

# Example usage
tensor = base_model_tensors['fan']['vision']  # Your input tensor
reduced_embedding = reduce_dimensions(tensor)
```

```
----------------------------------------------------------------------
-----
ModuleNotFoundError                       Traceback (most recent call
last)
Cell In[9], line 2
      1 import torch
----> 2 import umap
      3 import matplotlib.pyplot as plt
      5 def reduce_dimensions(tensor):
      6     # Ensure tensor is 2D (samples x features)

ModuleNotFoundError: No module named 'umap'
```