**Prompt Engineering:**

* write a sample code for a unet, with scripts for the data loader, unet, training procedure, test and validate, and how to run it
* Write a complete example in Python for implementing a U-Net architecture for semantic segmentation using PyTorch. Include scripts for data loading, defining the U-Net model, the training procedure, validation, testing, and how to run the code. The model should be optimized for binary masks
* Write a complete example in Python for implementing a U-Net architecture for semantic segmentation using PyTorch. The input images are in grayscale, and the masks are binary (containing 0 for background and 1 for the object). Include scripts for data loading, defining the U-Net model, the training procedure, validation, testing, and how to run the code. The model should be optimized for binary segmentation.
* Write a complete ready-to-run implementation in Python for implementing a U-Net architecture for semantic segmentation using PyTorch. The input images are in grayscale, and the masks are binary. Include scripts for data loading, defining the U-Net model, and the training procedure. The model should be optimized for binary segmentation
* Please provide a complete ready-to-run Python implementation for a U-Net architecture using PyTorch for binary segmentation of grayscale images. The dataset should consist of grayscale images and binary masks. The code should be structured into importable scripts with the following:

1. Required library imports.
2. A custom Dataset class for loading the grayscale images and corresponding binary masks.
3. A UNet model class definition using PyTorch, optimized for binary segmentation.
4. Functions for the training procedure, validation procedure, and testing procedure.
5. A function to split the dataset into 80% training, 10% validation, and 10% testing sets using train\_test\_split.
6. DataLoader setup for training, validation, and testing sets.
7. A function to visualize the predicted masks, the input image, and the ground truth during testing.
8. All the above functionalities should be implemented in a modular way so that they can be imported as scripts or classes.
9. Ensure that everything can be executed in an if \_\_name\_\_ == "\_\_main\_\_": block with instructions for loading the dataset, configuring hyperparameters, and saving the trained model.

* Please provide a complete ready-to-run Python implementation for a U-Net architecture using PyTorch for binary segmentation of grayscale images. The dataset should consist of grayscale images and binary masks. The code should be structured into importable scripts with the following:

1. Required library imports. A custom Dataset class for loading the grayscale images and corresponding binary masks.
2. A UNet model class definition using PyTorch, optimized for binary segmentation. Functions for the training procedure, validation procedure, and testing procedure.
3. A function to split the dataset into 80% training, 10% validation, and 10% testing sets using train\_test\_split.
4. DataLoader setup for training, validation, and testing sets.
5. A function to visualize the predicted masks, the input image, and the ground truth during testing.
6. All the above functionalities should be implemented in a modular way so that they can be imported as scripts or classes.
7. Ensure that everything can be executed in an if \_\_name\_\_ == "\_\_main\_\_": block with instructions for loading the dataset, configuring hyperparameters, and saving the trained model.
8. the directory structure should be:

unet\_segmentation/

├── dataset.py

├── model.py

├── train.py

├── main.py

* Please provide a complete ready-to-run Python implementation for a U-Net architecture using PyTorch, for binary segmentation of grayscale images. The dataset should consist of grayscale images and binary masks, which are located in the same path (the images are named with numbering like "number.png" and the masks "number\_seg.png).

1. The code should be structured into importable scripts with the following:
2. A custom Dataset class for loading the grayscale images and corresponding binary masks.
3. A UNet model class definition using PyTorch, optimized for binary segmentation.
4. Functions for the training procedure, validation procedure, and testing procedure.
5. calculating and saving the dice scores during validation and testing, in a dataframe as an excel file.
6. A function to split the dataset into 80% training, 10% validation, and 10% testing sets using train\_test\_split.
7. DataLoader setup for training, validation, and testing sets.
8. A function to visualize the predicted masks, the input image, and the ground truth during testing (subplots for 5 random samples in one plot, save the plot as png)
9. All the above functionalities should be implemented in a modular way so that they can be imported as scripts or classes.
10. Ensure that everything can be executed in an if \_\_name\_\_ == "\_\_main\_\_": block with instructions for loading the dataset, configuring hyperparameters, and saving the trained model.
11. the directory structure should be:

unet\_segmentation/

├── dataset.py

├── model.py

├── train.py

├── main.py

* Please provide a complete ready-to-run Python implementation for a U-Net architecture using PyTorch, for binary segmentation of grayscale images. The dataset should consist of grayscale images and binary masks, which are located in the same path (the images are named with numbering like "number.png" and the masks "number\_seg.png). The code should be structured into importable scripts with the following:

1. A custom Dataset class for loading the grayscale images and corresponding binary masks. - A UNet model class definition using PyTorch, optimized for binary segmentation.
2. Functions for the training procedure, validation procedure, and testing procedure.
3. during validation, dice scores should be calculated and saved in a dataframe as an excel file in a given path. also during tetsing.
4. the progress and info should be shown epoch by epoch using tgdm progress bars, during training, validation and testing.
5. training duration should be timed and printed
6. A function to split the dataset into: 80% training, 10% validation, 10% test sets using train\_test\_split. - DataLoader setup for training, validation, and testing sets.
7. A function to visualize the predicted masks, the input image, and the ground truth during testing (subplots for 5 random samples in one plot, save the plot as png)
8. All the above functionalities should be implemented in a modular way so that they can be imported as scripts or classes. Ensure that everything can be executed in an if \_\_name\_\_ == "\_\_main\_\_": block with instructions for loading the dataset, configuring hyperparameters, and saving the trained model.
9. the directory structure should be:

unet\_segmentation/

├── dataset.py

├── model.py

├── train.py

├── main.py

* Please provide a complete ready-to-run Python implementation for a U-Net architecture using PyTorch, for binary segmentation of grayscale images. The dataset should consist of grayscale images and binary masks, which are located in the same path (the images are named with numbering like "number.png" and the masks "number\_seg.png) The code should be structured into importable scripts with the following:

1. A custom Dataset class for loading the grayscale images and corresponding binary masks.
2. A UNet model class definition using PyTorch, optimized for binary segmentation.
3. Functions for the training procedure, validation procedure, and testing procedure.
4. during validation, dice scores should be calculated and saved in a dataframe as an excel file in a given path. also during tetsing.
5. the progress and info should be shown epoch by epoch using tgdm progress bars, during training, validation and testing.
6. training duration should be timed and printed
7. additionally, after the training and validation are done, i want you to visualize training and validation losses per epochs (on the same plot) and save the plot to the given path as .png. (so the loss values need to be appended for plotting at the end)
8. A function to split the dataset into: 80% training, 10% validation, 10% test sets using train\_test\_split.
9. DataLoader setup for training, validation, and testing sets.
10. A function to visualize the predicted masks, the input image, and the ground truth during testing (subplots for 5 random samples in one plot, save the plot as png)
11. All the above functionalities should be implemented in a modular way so that they can be imported as scripts or classes.
12. Ensure that everything can be executed in an if \_\_name\_\_ == "\_\_main\_\_": block with instructions for loading the dataset, configuring hyperparameters, and saving the trained model..
13. the directory structure should be: unet\_segmentation/

├── dataset.py

├── model.py

├── train.py

├── main.py

* Please provide a complete ready-to-run Python implementation for a U-Net architecture using PyTorch, for binary segmentation of grayscale images. The dataset should consist of grayscale images and binary masks, which are located in the same path (the images are named with numbering like "number.png" and the masks "number\_seg.png"). The code should be structured into importable scripts with the following:

1. A custom Dataset class for loading the grayscale images and corresponding binary masks.
2. A UNet model class definition using PyTorch, optimized for binary segmentation.
3. Functions for the training procedure, validation procedure, and testing procedure, considering following requirements:
4. Compute training and validation losses.
5. A function to visualize training losses and validation losses over epochs afterwards, on the same plot. Save plot in a specified save path as png.
6. During Validation: Dice scores should be calculated for each batch, and all dice coefficients for the entire epoch should be stored in an Excel file, where each row in the Excel file should represent an epoch, and each column should represent a batch. The Excel file (named “validation\_dice\_scores” should be saved at the specified save path after validation is done.
7. Do the same for the testing and save the excel file as “test\_dice\_scores” after testing is done.
8. The progress and info should be shown epoch by epoch using `tqdm` progress bars during training, validation, and testing.
9. Training duration should be timed and printed.
10. A function to visualize the predicted masks, the input image, and the ground truth during testing (subplots for 5 random samples in one plot, in 5 rows and 3 columns (Input Image, Ground Truth, Prediction) save the plot as `.png`).
11. A function to split the dataset into: 80% training, 10% validation, and 10% test sets using `train\_test\_split`.
12. DataLoader setup for training, validation, and testing sets.
13. All the above functionalities should be implemented in a modular way so that they can be imported as scripts or classes.
14. Ensure that everything can be executed in an `if \_\_name\_\_ == "\_\_main\_\_":` block with instructions for loading the dataset, configuring hyperparameters, and saving the trained model.
15. The directory structure should be:

unet\_segmentation/

├── dataset.py

├── model.py

├── train.py

├── main.py

* Please provide a complete ready-to-run Python implementation for a U-Net architecture using PyTorch, for binary segmentation of grayscale images. The dataset should consist of grayscale images and corresponding binary masks, which are located in the same path (the images are named with numbering like "number.png" and the masks "number\_seg.png"). The code should be structured into importable scripts with the following:

1. A custom Dataset class for loading the grayscale images and corresponding binary masks.
2. A UNet model class definition using PyTorch, optimized for binary segmentation.
3. Functions for the training procedure, validation procedure, and testing procedure.
4. Compute training and validation losses.
5. Training and Validation Losses: Track and save the average training loss for each epoch in a separate Excel file (train\_losses.xlsx) in the save path. The first row should contain the epoch numbers, and the second row should contain the corresponding average training loss for each epoch.
6. Similarly, save the validation loss for each epoch in an Excel file (val\_losses.xlsx) with the same structure.
7. At the end of training, save both the entire model and the model's state dictionary (.pth files) in the save path
8. A function to visualize training losses and validation losses over epochs afterwards, on the same plot. Save plot in a specified save path as png.
9. During Validation: Dice scores should be calculated for each batch, and all dice coefficients for the entire epoch should be stored in an Excel file, where each row in the Excel file should represent an epoch, and each column should represent a batch. The Excel file (named “validation\_dice\_scores” should be saved at the specified save path after validation is done.
10. Do the same for the testing and save the excel file as “test\_dice\_scores” after testing is done.
11. The progress and info should be shown epoch by epoch using `tqdm` progress bars during training, validation, and testing.
12. Training duration should be timed and printed.
13. A function to visualize the predicted masks, the input image, and the ground truth during testing (subplots for 5 random samples in one plot, in 5 rows and 3 columns (Input Image, Ground Truth, Prediction) save the plot as `.png`).
14. A function to split the dataset into: 80% training, 10% validation, and 10% test sets using `train\_test\_split`.
15. DataLoader setup for training, validation, and testing sets.
16. All the above functionalities should be implemented in a modular way so that they can be imported as scripts or classes.
17. Ensure that everything can be executed in an `if \_\_name\_\_ == "\_\_main\_\_":` block with instructions for loading the dataset, configuring hyperparameters, and saving the trained model.

The directory structure should be:

unet\_segmentation/

├── dataset.py

├── model.py

├── train.py

├── main.py

* Please provide a complete ready-to-run Python implementation for a U-Net architecture using PyTorch, for binary segmentation of grayscale images. The dataset should consist of grayscale images and corresponding binary masks, which are located in the same path (the images are named with numbering like "number.png" and the masks "number\_seg.png", around 5000 samples in total). It should work and provide correct predictions. The code should be structured into importable scripts with the following:

1. A custom Dataset class for loading the grayscale images and corresponding binary masks. (only the png files, ignore meta files or additional files)
2. A UNet model class definition using PyTorch, optimized for binary segmentation (with small foreground, mostly background)
3. Functions for the training procedure, validation procedure, and testing procedure.

**Considerations:**

1. Ensure needed transforms on input image and mask pairs (Re-size images correctly, ensure gray scale and binary…)
2. Avoid overfitting and data leakage (without using augmentations), choose appropriate learning rate, batch size, etc

**For Training and Validation:**

1. Compute training and validation losses.
2. Track and save the average training loss for each epoch in a separate Excel file (train\_losses.xlsx) in a given save path. The first row should contain the epoch numbers, and the second row should contain the corresponding average training loss for each epoch.
3. Similarly, save the validation loss for each epoch in an Excel file (val\_losses.xlsx) with the same structure.
4. At the end of training, save both the entire model and the model's state dictionary (.pth files) in the save path
5. Calculate total training time from start to end, and print at the end of training
6. The progress, phase (train/validation), and all info should be shown epoch by epoch using `tqdm` progress bars during training, validation, and testing.
7. A function to visualize training losses and validation losses over epochs afterwards, on the same plot. Save plot in a specified save path as png.
8. During Validation: Dice scores should be calculated for each batch, and all dice coefficients for the entire epoch should be stored in an Excel file, where each row in the Excel file should represent an epoch, and each column should represent a batch. The Excel file (named “validation\_dice\_scores” should be saved at the specified save path after validation is done.
9. Do the same for the testing and store dice the Dice scores the same way and save the excel file as “test\_dice\_scores” after testing is done.

**For testing:**

1. A function to visualize the predicted masks, the input image, and the ground truth, for 5 random samples, after testing is done. (subplots for 5 random samples in one plot, in 5 rows and 3 columns (Input Image, Ground Truth, Prediction). Display image file name /id above each. save the plot as `.png`).

**For Main:**

1. split the dataset into: 80% training, 10% validation, and 10% test sets using `train\_test\_split`. Print the sizes of each (how many images)
2. DataLoader setup for training, validation, and testing sets.
3. Ensure correct data loading, print data loader sizes.
4. All the above functionalities should be implemented in a modular way so that they can be imported as scripts or classes.
5. All visualizations should just be saved as png. No need to display the plots during running.
6. Ensure that everything can be executed in an `if \_\_name\_\_ == "\_\_main\_\_":` block with instructions for loading the dataset, configuring hyperparameters, and saving the trained model.

The directory structure should be:

unet\_segmentation/

├── dataset.py

├── model.py

├── train.py

├── main.py

* Please provide a complete ready-to-run Python implementation for a U-Net architecture using PyTorch, for binary segmentation of grayscale images. The dataset should consist of grayscale images and corresponding binary masks, which are located in the same path (the images are named with numbering like "number.png" and the masks "number\_seg.png", around 5000 samples in total). It should work and provide correct predictions. The code should be structured into importable scripts with the following, add all the noted considerations:

1. A custom Dataset class for loading the grayscale images and corresponding binary masks. (only the png files, ignore meta files or additional files in the directory)
2. A UNet model class definition using PyTorch, optimized for binary segmentation (with small foreground, mostly background)
3. Functions for the training procedure, validation procedure, and testing procedure.

**Considerations:**

1. Ensure needed transforms on input image and mask pairs (Re-size images correctly, ensure gray scale and binary…)
2. Avoid overfitting and data leakage (without using augmentations), choose appropriate learning rate, batch size, etc

**For Training and Validation:**

1. Compute training and validation losses.
2. Track and save the average training loss for each epoch in a separate Excel file (train\_losses.xlsx) in a given save path. The first row should contain the epoch numbers, and the second row should contain the corresponding average training loss for each epoch.
3. Similarly, save the validation loss for each epoch in an Excel file (val\_losses.xlsx) with the same structure.
4. At the end of training, save both the entire model and the model's state dictionary (.pth files) in the save path
5. Calculate total training time from start to end, and print at the end of training
6. The progress, phase (train/validation), and all info should be shown epoch by epoch using `tqdm` progress bars during training, validation, and testing.
7. A function to visualize training losses and validation losses over epochs afterwards, on the same plot. Save plot in a specified save path as png.
8. During Validation: Dice scores should be calculated for each batch, and all dice coefficients for the entire epoch should be stored in an Excel file, where each row in the Excel file should represent an epoch, and each column should represent a batch. The Excel file (named “validation\_dice\_scores” should be saved at the specified save path after validation is done.
9. Do the same for the testing and store dice the Dice scores the same way and save the excel file as “test\_dice\_scores” after testing is done.

**For testing:**

1. After Testing: A function to visualize the predicted masks, the input image, and the ground truth, for 5 random samples, after testing is done. (subplots for 5 random samples in one plot, in 5 rows and 3 columns (title at top: Input Image, Ground Truth, Prediction). Also display image file name /id string above each image and mask. save the plot as `.png`).

**For Main:**

1. Shuffle the data, split the dataset into: 80% training, 10% validation, and 10% test sets using `train\_test\_split`. Print the sizes of each (how many images), and copy the splitted data (images and their respective masks)
2. DataLoader setup for training, validation, and testing sets.
3. Ensure correct data loading, print data loader sizes.
4. All the above functionalities should be implemented in a modular way so that they can be imported as scripts or classes.
5. All visualizations should just be saved as png. No need to display the plots during running.
6. Ensure that everything can be executed in an `if \_\_name\_\_ == "\_\_main\_\_":` block with instructions for loading the dataset, configuring hyperparameters, and saving the trained model.

The directory structure should be:

unet\_segmentation/

├── dataset.py

├── model.py

├── train.py

├── main.py

* Please provide a complete ready-to-run Python implementation for a U-Net architecture using PyTorch, for binary segmentation of grayscale images. The dataset should consist of grayscale images and corresponding binary masks, which are located in the same path (the images are named with numbering like "number.png" and the masks "number\_seg.png", around 5000 samples in total). It should work and provide correct predictions, and save the losses and dice scores as instructed below. The code should be structured into importable scripts with the following:

1. A custom Dataset class for loading the grayscale images and corresponding binary masks. (only the png files, ignore meta files or additional files in the directory)
2. A UNet model class definition using PyTorch, optimized for binary segmentation (with small foreground, mostly background)
3. Functions for the training procedure, validation procedure, and Testing procedure.

**Considerations for each part:**

1. Ensure needed transforms on input image and mask pairs (Re-size images correctly, ensure gray scale and binary…)
2. Avoid overfitting and data leakage (without using augmentations), choose appropriate hyperparameters, learning rate, batch size, etc

* **For Training and Validation:**

1. Compute training and validation losses.
2. Track and save the average training loss for each epoch in a separate Excel file (train\_losses.xlsx) in a given save path. The first row should contain the epoch numbers, and the second row should contain the corresponding average training loss for each epoch.
3. Similarly, save the validation loss for each epoch in an Excel file (val\_losses.xlsx) with the same structure.
4. At the end of training, save both the entire model and the model's state dictionary (.pth files) in the save path
5. Calculate total training time from start to end, and print at the end of training
6. The progress, phase (train/validation), and all info should be shown epoch by epoch using `tqdm` progress bars during training, validation, and testing.
7. A function to visualize training losses and validation losses over epochs afterwards, on the same plot. Save plot in a specified save path as png.
8. During Validation: Dice scores should be calculated for each batch, and all dice coefficients for the entire epoch should be stored in an Excel file, where each row in the Excel file should represent an epoch, and each column should represent a batch. The Excel file (named “validation\_dice\_scores” should be saved at the specified save path after validation is done.
9. Do the same for the Testing and store dice the Dice scores the same way and save the excel file as “test\_dice\_scores” after testing is done.

* **For Testing:**

1. A function to visualize the predicted masks, the input image, and the ground truth, for 5 random samples, after testing is done. (subplots for 5 random samples in one plot, in 5 rows and 3 columns (title at top: Input Image, Ground Truth, Prediction). Also display image file name /id string above each image and mask. save the plot as `.png`).

* **For Main:**

1. Shuffle the data, split the dataset into: 80% training, 10% validation, and 10% test sets using `train\_test\_split`. Print the sizes of each (how many images), and copy the splitted data (images and their respective masks)
2. DataLoader setup for training, validation, and testing sets.
3. Ensure correct data loading, print data loader sizes.
4. All the above functionalities should be implemented in a modular way so that they can be imported as scripts or classes.
5. All visualizations should just be saved as png. No need to display the plots during running.
6. Ensure that everything can be executed in an `if \_\_name\_\_ == "\_\_main\_\_":` block with instructions for loading the dataset, configuring hyperparameters, and saving the trained model.

The directory structure should only be:

unet\_segmentation/

├── dataset.py

├── model.py

├── train.py

├── main.py

* Please provide a complete ready-to-run Python implementation for a U-Net architecture using PyTorch, for binary segmentation of grayscale images. The dataset should consist of grayscale images and corresponding binary masks, which are located in the same path (the images are named with numbering like "number.png" and the masks "number\_seg.png", around 5000 samples in total). It should work and provide correct predictions, and save the losses and dice scores as instructed below. The code should be structured into importable scripts with the following:

1. A custom Dataset class for loading the grayscale images and corresponding binary masks. (only the png files, ignore meta files or additional files in the directory)
2. A UNet model class definition using PyTorch, optimized for binary segmentation (with small foreground, mostly background)
3. Functions for the training procedure, validation procedure, and Testing procedure.

**Considerations for each part:**

1. Ensure needed transforms on input image and mask pairs (Re-size images correctly, ensure gray scale and binary…)
2. Avoid overfitting and data leakage (without using augmentations), choose appropriate hyperparameters, learning rate, batch size, etc

* **For Training and Validation:**

1. Compute training and validation losses.
2. Track and save the average training loss for each epoch in a separate Excel file (train\_losses.xlsx) in a given save path. The first row should contain the epoch numbers, and the second row should contain the corresponding average training loss for each epoch.
3. Similarly, save the validation loss for each epoch in an Excel file (val\_losses.xlsx) with the same structure.
4. At the end of training, save both the entire model and the model's state dictionary (.pth files) in the save path
5. Calculate total training time from start to end, and print at the end of training
6. The progress, phase (train/validation), and all info should be shown epoch by epoch using `tqdm` progress bars during training, validation, and testing.
7. A function to visualize training losses and validation losses over epochs afterwards, on the same plot. Save plot in a specified save path as png.
8. During Validation: Dice scores should be calculated for each batch, and all dice coefficients for the entire epoch should be stored in an Excel file, where each row in the Excel file should represent an epoch, and each column should represent a batch. The Excel file (named “validation\_dice\_scores” should be saved at the specified save path after validation is done.
9. Do the same for the Testing and store dice the Dice scores the same way and save the excel file as “test\_dice\_scores” after testing is done.

* **For Testing:**

1. A function to visualize the predicted masks, the input image, and the ground truth, for 5 random samples, after testing is done. (subplots for 5 random samples in one plot, in 5 rows and 3 columns (title at top: Input Image, Ground Truth, Prediction). Also display image file name /id string above each image and mask. save the plot as `.png`).

* **For Main:**

1. Shuffle the data, split the dataset into: 80% training, 10% validation, and 10% test sets using `train\_test\_split`. Print the sizes of each (how many images), and copy the splitted data (images and their respective masks)
2. DataLoader setup for training, validation, and testing sets.
3. Ensure correct data loading, print data loader sizes.
4. Also print model summary, and total number of model parameters (n learnable params)
5. All the above functionalities should be implemented in a modular way so that they can be imported as scripts or classes.
6. All visualizations should just be saved as png. No need to display the plots during running.
7. Ensure that everything can be executed in an `if \_\_name\_\_ == "\_\_main\_\_":` block with instructions for loading the dataset, configuring hyperparameters, and saving the trained model.

The directory structure should only be:

unet\_segmentation/

├── dataset.py

├── model.py

├── train.py

├── main.py