Example:

Prefix :

```
import os
import gdown
url = 'https://drive.google.com/uc?
id=1JLxhdIddq6_vKlHml7jT48VaeXoJjvpR'
output_filename = 'data.zip'
# Get current working directory
cwd = os.getcwd()
# Concatenate current working directory and output filename
output_path = os.path.join(cwd, output_filename)
# Download file from Google Drive to output path
gdown.download(url, output_path, quiet=False)
import zipfile
with zipfile.ZipFile(output_filename, 'r') as zip_ref:
```

Actual Middle :

```
    zip_ref.extractall('./data')
```

Generated Middle :

```
    zip_ref.extractall('./')
```

Suffix :

In this example, how can the model knows about the file name “./data”, so I would assume it had done a great job, in determining the line but cant blame it for the file name.

Example:

Prefix:
```
import os
import gdown
url = 'https://drive.google.com/uc?
id=1JLxhdIddq6_vKlHml7jT48VaeXoJjvpR'
```

Actual Middle:
```
output_filename = 'data.zip'
# Get current working directory
cwd = os.getcwd()
# Concatenate current working directory and output filename
```

Generated:
```
cwd = os.getcwd()
print("Current working directory :", cwd)
output_filename = "model.zip"
```

Suffix:
```
output_path = os.path.join(cwd, output_filename)
# Download file from Google Drive to output path
gdown.download(url, output_path, quiet=False)
import zipfile
with zipfile.ZipFile(output_filename, 'r') as zip_ref:
    zip_ref.extractall('./data')
```

In this example, the model has seen it needs to define output_filename and cwd because it will be used in the suffix, so it had defined both correctly.

Even if output_filename has different name but I cant say this is wrong as we had the whole free to define it as whatever we want.

Example :

Prefix:

```
import os
import gdown
url = 'https://drive.google.com/uc?
id=1JLxhdIddq6_vKlHml7jT48VaeXoJjvpR'
output_filename = 'data.zip'
# Get current working directory
cwd = os.getcwd()
```

Actual Middle:

```
# Concatenate current working directory and output filename
output_path = os.path.join(cwd, output_filename)
```

Generated Middle:

```
print("Current working dir is " + cwd)
# Create a folder named data in the current working directory if
it does not exist
if not os.path.exists('./data'):
    print("Creating new folder")
    os.makedirs('./data')
else:
    print("Folder already exists.")
# Set output path for downloaded file
output_path = os.path.join(os.getcwd(), output_filename)
print("Output path is " + output_path)
```

Suffix:

```
# Download file from Google Drive to output path
gdown.download(url, output_path, quiet=False)
import zipfile
with zipfile.ZipFile(output_filename, 'r') as zip_ref:
    zip_ref.extractall('./data')
```

Here, the generated middle does much better job than the actual middle, because it actually adds a new check “Create a folder named data in the current working directory if it does not exist” unlike the actual middle code.

The generated code did so, because in the suffix, we actually zip something into data folder without any check on whether this file is really here or not.

Example :

Prefix:
```
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing.image import
ImageDataGenerator
from keras.models import Sequential
from keras.layers import Dense
from keras.utils import to_categorical
from keras import regularizers
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from skimage.transform import resize
from skimage.measure import regionprops
from skimage.color import label2rgb
from skimage.morphology import closing, disk, skeletonize
from skimage.util import invert
from skimage.segmentation import clear_border
from skimage.feature import hog, local_binary_pattern
from skimage.color import rgb2gray
from skimage.feature import daisy

from scipy.fftpack import fft
from scipy.signal import convolve2d
from scipy.ndimage import convolve
from scipy import ndimage as ndi
```

Actual Middle:
```
from pathlib import Path
from PIL import Image, ImageOps
```

Generated Middle:

Suffix:
```
from pyefd import elliptic_fourier_descriptors
# from multiprocessing import Pool
from colorama import Fore, Back, Style
from skfuzzy.cluster import cmeans
```

This file was called imports.py, where we just imported some modules. Here Generated middle returned empty string. Actually this is very understandable, because first of all how can it predicts the imports you need without seeing the code. Second thing I noted, that the actual middle itself has no relation to suffix and prefix, it imported from 2 different modules so there is no correlation or pattern here.
We could have a problem if the suffix and prefix are from the same module example skimage so the middle one could also import some stuff from the skimage but thats not the case here

Example:

```
Prefix:
A lot of code … …
for i, filename in enumerate(files):
    img = cv2.imread(os.path.join(path, filename))
    img = np.array(img)
    # Get current time
    start = time.perf_counter()

Actual Middle:
    # Resize the image
    img = data_loader.custom_resize_img(img)
    # Illumination Preprocessing
    illuminated_test, _ = illumination_processing.process_image(img)
    # Image Alignment
    aligned_test = image_aligner.align_image([illuminated_test])[0]

Generated Middle:
 # Align images
    aligned_test = image_aligner.align_images([img],
data_loader.get_reference())[0]

Suffix:
    # Feature extraction and selection
    daisy_features_test =
feature_extractor.extract_daisy_features([aligned_test])[0]

    pca_daisy_features_test =
feature_selector.test_pca(daisy_features_test,pca)

    pca_daisy_features_test = (pca_daisy_features_test -
extracted_features_train_mean) /extracted_features_train_std
```

In this file , the generated middle misses a whole step in the pipeline of the image proccessing we have which is Illumination Preprocessing, **But** actually we can't do anything here because this is optional step to improve the illumination in images processed. So by checking the example in details you will understand its ok, its not that bad. At least it knew that we had aligned_test from the suffix and it needs to do the alignment.

And big advantage is it used the image_aligner class that we had already, so it really gave importance to the prefix, and didnt try to implement the alignment itself.

Example

```
Prefix:
Lots of code … …
   aligned_test = image_aligner.align_image([illuminated_test])[0]
    # Feature extraction and selection
    daisy_features_test =
feature_extractor.extract_daisy_features([aligned_test])[0]
    pca_daisy_features_test =
feature_selector.test_pca(daisy_features_test,pca)
    pca_daisy_features_test = (pca_daisy_features_test -
extracted_features_train_mean) /extracted_features_train_std


Actual Middle:
     # Model loading and prediction
    model_prediction = model.predict(pca_daisy_features_test)
    # Only in case of ANN
    model_prediction = model_prediction.argmax(axis=1)
    # stop timer
    end = time.perf_counter()
    total_time_seconds = round(end - start, 3)


Generated Middle:
 # Predicting the class label
    model_prediction =
model.predict(np.expand_dims(pca_daisy_features_test, axis=0))
    total_time_seconds = round((time.perf_counter()-start), 6)

Suffix:
    # write the prediction in results file
    results_file.write(f"{int(model_prediction[0])}\n")
    # write the time in times file
    time_file.write(f"{total_time_seconds}\n")
results_file.close()
time_file.close()
```

In this file, The generated Middle actually had compressed some of the lines and made the whole instruction inline example in the total_time_seconds it calculated time.perf_counter inline.

One other thing, it did expand_dims of the numpy array.
It also missed doing the argmax along the model prediction, but actually the problem here that the model used was not defined in the same file, so it cant know how the output is structured.

Example

```
Prefix:
Lots of code … …
# stop timer
    end = time.perf_counter()
    total_time_seconds = round(end - start, 3)
    # write the prediction in results file
    results_file.write(f"{int(model_prediction[0])}\n")
    # write the time in times file

Actual Middle:
     time_file.write(f"{total_time_seconds}\n")


Generated Middle:
 time_file.write(f"{total_time_seconds}\n")

Suffix:
results_file.close()
time_file.close()
```

In this relatively easy example, we got exact match of the output !!