Generating the dataset files for your project involves several steps, including extracting features from PE (Portable Executable) files such as opcodes, strings, and hashing word sequences. I'll walk you through the process of generating the necessary dataset files.

Steps to Generate Dataset Files

Here's a high-level breakdown of the steps required to generate the dataset files:

1. **Extract Opcode Sequences from PE Files**
2. **Extract Strings from PE Files**
3. **Hash the Strings and Convert to Images**
4. **Create the Dataset Files**

You can use Python libraries like **pefile**, **Capstone**, and **TensorFlow** to achieve this. I'll provide you with code snippets for each step.

---

1. Extract Opcode Sequences from PE Files

We need to extract the opcodes from the disassembled binary code of PE files. This can be done using **Capstone**, which is a disassembly framework.

Code to Extract Opcodes:

```
import pefile
from capstone import Cs

def extract_opcodes(pe_path):
    try:
        pe = pefile.PE(pe_path, fast_load=True)
        entry_point = pe.OPTIONAL_HEADER.AddressOfEntryPoint
        data = pe.get_memory_mapped_image()[entry_point:]

        cs = Cs(CS_ARCH_X86, CS_MODE_32)
        opcodes = []

        for i in cs.disasm(data, 0x1000):
            opcodes.append(i.mnemonic)

        return opcodes

    except Exception as e:
        print(f"Error processing file {pe_path}: {str(e)}")
        return []

# Example usage:
pe_file_path = "path_to_pe_file.exe"
opcodes = extract_opcodes(pe_file_path)
print(opcodes)
```

This function will return a list of opcode mnemonics from the PE file.

---

2. Extract Strings from PE Files

We need to extract printable strings from the PE files. This can be done using a simple Python script.

Code to Extract Strings:

```
import string

def extract_strings(pe_path, min_length=4):
    try:
        with open(pe_path, "rb") as f:
            data = f.read()

        result = ""
        for c in data.decode(errors="ignore"):
            if c in string.printable:
                result += c
            elif len(result) >= min_length:
                yield result
                result = ""
        if len(result) >= min_length:  # catch the last string
            yield result

    except Exception as e:
        print(f"Error processing file {pe_path}: {str(e)}")

# Example usage:
pe_file_path = "path_to_pe_file.exe"
strings = list(extract_strings(pe_file_path))
print(strings)
```

This function will return a list of strings extracted from the PE file, where each string has at least `min_length` printable characters.

---

3. Hash the Strings and Convert to Images

After extracting the strings, we hash them using the **hashing trick** to generate integer representations. We then reshape these hashes into 100x100 grayscale images for the string model.

Code to Hash Strings:

```
import numpy as np
from tensorflow.keras.preprocessing.text import hashing_trick

def hash_strings_to_image(strings, vocab_size, max_seq_len=10000):
    hashed_seqs = []

    for string in strings:
        hashed_ids = hashing_trick(string, vocab_size, hash_function='md5')

        # Make sure the length matches the max sequence length
        if len(hashed_ids) < max_seq_len:
```

```
            hashed_ids += [0] * (max_seq_len - len(hashed_ids))

        # Reshape to a 100x100 image
        image = np.array(hashed_ids[:10000]).reshape(100, 100, 1)
        hashed_seqs.append(image)

    return np.array(hashed_seqs)

# Example usage:
strings = ["malware", "file", "example", "with", "strings"]
vocab_size = 10000  # Set this according to your dataset
hashed_images = hash_strings_to_image(strings, vocab_size)
print(hashed_images.shape)  # Expected output: (num_strings, 100, 100, 1)
```

This function will return a numpy array of images, each representing a sequence of hashed strings.

---

4. Create the Dataset Files

We now combine the **opcode sequences** and **string images** for each PE file. We will store these combined features along with the labels (benign, malware, ransomware) in numpy files.

Code to Generate Dataset:

```
import numpy as np
import os

def generate_dataset(pe_file_paths, labels, vocab_size=10000):
    opcodes_data = []
    strings_data = []
    labels_data = []

    for idx, pe_file in enumerate(pe_file_paths):
        print(f"Processing {pe_file}...")

        # Extract opcodes
        opcodes = extract_opcodes(pe_file)
        opcodes_freq_vector = [0] * 50  # 50 most common opcodes
        for i, opcode in enumerate(opcodes[:50]):  # Limit to top 50
opcodes
            opcodes_freq_vector[i] = round((opcodes.count(opcode) /
len(opcodes)) * 100, 2)

        opcodes_data.append(np.array(opcodes_freq_vector))

        # Extract strings and hash to image
        strings = list(extract_strings(pe_file))
        hashed_image = hash_strings_to_image(strings, vocab_size)
        strings_data.append(hashed_image[0])  # Assume one image per PE
file

        # Label (benign, malware, ransomware)
        labels_data.append(labels[idx])

    # Save dataset as numpy arrays
    np.save('opcode_features.npy', np.array(opcodes_data))
```

```
    np.save('string_features.npy', np.array(strings_data))
    np.save('labels.npy', np.array(labels_data))

    print("Dataset generation complete.")

# Example usage:
pe_file_paths = ["path_to_pe_file1.exe", "path_to_pe_file2.exe"]  # List of
PE file paths
labels = ['malware', 'benign']  # Corresponding labels for each file
generate_dataset(pe_file_paths, labels)
```

Generated Dataset Files:

1. **opcode_features.npy**: Contains the frequency vectors of opcodes (shape: `num_samples x 50`).
2. **string_features.npy**: Contains the hashed string images (shape: `num_samples x 100 x 100 x 1`).
3. **labels.npy**: Contains the labels (`benign`, `malware`, `ransomware`) for each PE file (shape: `num_samples`).

Conclusion

After running the code snippets, you will have the necessary dataset files for training the models in your project. Here's what you'll have:

1. **`opcode_features.npy`**: The opcode frequency vectors for each PE file.
2. **`string_features.npy`**: The string images (grayscale) for each PE file.
3. **`labels.npy`**: The labels corresponding to each PE file (benign, malware, or ransomware).

These files can be used for training and testing the models described in your project.