The python script is a pipeline consisting of a series of Stages from processes the HEALSL data to analyzing results. The Stages are as follows:

Stage 1 Preprocessing

This stage is responsible for gathering only the necessary features into a common dataset. Since the features required are scattered in three files: questionnaire, age, and narrative, the script first examines a predefined directory to look for relevant data files, then extracts and merge the features into one dataset. The columns required are ‘open\_narrative’, ‘sex\_cod’, ‘age\_value\_death’, and ‘age\_unit\_death’, joint by ‘rowid’ as the key. This process is repeated for age groups: neonate, child, and adult, and for round: one and two. The combined dataset has 11,887 records, and we will refer to this as “full data”.

Next, is to create a sampling dataset. The purpose is to repeatedly send the same API request to evaluate how often similar results are produced. By default setting, the number of records to sample is 100 and each repeated ten times. The script will proportionally sample in respect to the total number of available records from each age group and round. This dataset has 1,000 records, and we will refer to this as “sample data”.

Stage 2 Getting API Response

This stage is responsible for generating GPT responses using the API. It involves importing the dataset generated by Stage 1, iterating through each record to compile a request package, sending the package through OpenAI API, receiving a response package from the API, and saving the results periodically.

Request Package

The Chat Completion object is utilized to facilitate the sending of a request package through OpenAI’s API, and to receive the corresponding response. This request package encapsulates the “prompts” and parameters needed to process each record, and the response package contains the output from the model and relevant details. Moreover, some setup, such as acquiring up an API key is required before OpenAI’s API will recognize the request package. The latest setup information can be found in the official OpenAI documentation.

There are two parts to Prompts: the system prompt, and the user prompt.

The system prompt provides the model with context on its role and expectations, defined in plain text. For this experiment, every record shared a similar system prompt, and we defined it as “You are a physician with expertise in determining underlying causes of death in Sierra Leone by assigning the most probable ICD-10 code for each death using verbal autopsy narratives. Return only the ICD-10 code without description. E.g. A00. If there are multiple ICD-10 codes, show one code per line.”

The user prompt provides specific instructions and individual record data, also defined in plain text. For this experiment, every record adheres to a predefined user prompt template, “Determine the underlying cause of death and provide the most probable ICD-10 code for a verbal autopsy narrative of a ‘AGE\_VALUE\_DEATH’ ‘AGE\_UNIT\_DEATH’ old ‘SEX\_COD’ death in Sierra Leone: ‘OPEN\_NARRATIVE’”. Uppercased text are variables to be filled based on each record. ‘AGE\_VALUE\_DEATH’ is the numeric age and ‘AGE\_UNIT\_DEATH’ is either “month” or “old”. ‘SEX\_COD’ is “male” or “female”. ‘OPEN\_NARRATIVE’ is the narrative for each record.

Other parameters for the request package include ‘model’, ‘temperature’, ‘logprobs’, and ‘max\_tokens’. The ‘model’ parameter specifies which model is to be used. To strive for consistency and reproducibility of the results, we used specific versions of the GPT-3 and GPT-4 models; gpt-3.5-turbo-0125 and gpt-4-0613, respectively. This differs from the aliases which may change over time as newer models are released by OpenAI. The ‘temperature’ parameter can be any value between 0 and 2. Low temperature values result in more deterministic responses, while high temperature values result to more random responses. For this experiment, the ‘temperature’ is set to 0. The ‘lobprobs’ parameter determines whether the response package should include the logarithmic probabilities for each output tokens. For this experiment, ‘logprobs’ is set to True. The ‘max\_token’ parameter sets a limitation on the number of output tokens, which helps with reducing excessive outputs and reduces wastage, since we are interested in up to five ICD-10 codes. For this experiment, ‘max\_token’ is set to 30, which should allow approximately six ICD-10 codes including room for margin.

Other non-essential parameters that may control the behaviour of Stage 2 are: DEMO prefixed parameters that reduces the input dataset to smaller subset of the original for demonstration or testing purposes, and DROP\_EXCESS\_COLUMNS which allows non-essential features to passthrough from the input to the output. By default, DROP\_EXCESS\_COLUMNS is set to False, since we want to retain ‘age\_group’ and ‘round’ features for later stages.

Save Point Mechanism

The Stage 2 process is time consuming as each record must go through the API one at a time. Since the API is pay-per-tokens being sent and received, premature termination of this process without saving the results would mean restarting from the beginning. Therefore, a save point mechanism has been implemented to periodically write API responses to file. To complement with this mechanism, at the very beginning of Stage 2, the script will look for and read any previous saved point to memory. If none is found, a blank storage is initialized. While iterating through each record, the script will check from memory whether an API response already exists and skip the record accordingly. The frequency of the save points is by default set every three records. While a lower value means saving more frequently resulting to smaller loss, as more records are processed, the increase in file size to write will result to longer write-to-disk time.

This concludes the Stage 2 process, and the aforementioned processes are executed on both full data and sample data by default.

Stage 3 ICD-10 Codes and Information Extraction

This stage examines the responses package acquired from Stage 2 and extract ICD-10 codes from the output message, combined logarithmic probabilities, tokens consumption, and other relevant information.

The simplest form of message output resides in message/content inside the response package. In most cases, the output is just an ICD-10 code in plain text. However, in some case, there may be more than one ICD-10 code or other irrelevant text—such as description that entails after the code. Hence, the script is responsible for identifying and extracting up to five valid ICD-10 codes from the output, and the remainder is disregarded. The ICD-10 codes are then presented in five new columns in the order they were extracted: ‘cause1\_icd10’, ‘cause2\_icd10’ to ‘cause5\_icd10’, where the first code becomes the primary cause of death predicted by the model.

The aforementioned processes in Step 3 are executed on full data and sample data by default.

Stage 4 Final touches

The final stage varies depending on whether full data and sample data is being processed.

For full data, the previously extracted ICD-10 codes are separated into individual files based on the ‘round’ column.

As for sample data, recall its purpose is to investigate the frequency of similar GPT responses when presented with the same prompt repeatedly. Specifically, 100 records are sampled from the full data, duplicated 10 times each.

To simplify the experiment, after Stage 3, we decided to retain only the primary ICD-10 code ‘cause1\_icd10’, and also only the code before the decimal point—discarding any subcategories or subclassifications after the decimal point. Then, we aggregate the 10 ICD-10 codes for each record and count the number of times the same ICD-10 code was produced. The results are presented in 10 columns, where ‘same\_cause1\_icd10\_1x’ denotes an ICD-10 code is repeatedly produced only once, ‘same\_cause1\_icd10\_2x’ denotes an ICD-10 code is repeatedly produced twice, up to ‘same\_cause1\_icd10\_10x’ denoting an ICD-10 code is repeatedly produced 10 times, out of the 10 times we had tested. Some columns may have values greater than one if there are multiple ICD-10 codes repeatedly produced fewer than five times.

Lastly, we repeat this process for CGHR-10 titles. After retaining only the ICD-10 code before the decimal point described in the previous paragraph, the ICD-10 code is converted to CGHR-10 titles using the mapping file icd10\_cghr10\_v1.csv that may be obtained from openmortality.org. In some rare cases, where the mapping of an ICD-10 to CGHR-10 is not listed in the mapping file, an “NA” value is filled in for those instances. Then, we follow a similar process of aggregating the CGHR-10 titles and counting the number of times the same title was produced. Results are presented in 10 columns, ‘same\_cause1\_cghr10\_1x’, ‘same\_cause1\_cghr10\_2x’ to ‘same\_cause1\_cghr10\_10x’, denoting the number of times a CGHR-10 title was repeatedly produced.

In FILEA.py, the model use