Prepared By Deivyansh Singh

Key Findings from Parakeet's Code

Dataset used for training:

1. Dataset represents a collection of construction-related products and pipe fittings along with their carbon emissions.
2. It is used to calculate the environmental impact due to the product.
3. It consists of product information, industry classification code, and carbon emission metrics
4. Quantifies the environmental impact of their material procurement and usage.

Structure of the Dataset:

"COUPLING BRASS COMP GJX COMPGJ 1 1-/4IN"

1. It consists of information about material and precise dimension.
2. It is used for emission factor mapping.
3. Each entry is associated with a NAICS code.
4. Environmental impact is denoted by "CO2e_per_dollar_final."
5. Iron and steel products have higher emissions, and PVC-related materials have the lowest.

Ways the dataset is created or generated:

1. Dataset is derived from existing environmental impact factor databases combined with human-validated recommendations.
2. It utilises two types of EFs:
    a. EEIO: Environmentally Extended Input-Output
    b. pLCA: process Life Cycle Assessment

Contents of the dataset:

1. Business activity description or product specifications that need EF mapping.
2. Corresponding appropriate EFs for standardized datasets.
3. Human annotations or validations that establish ground truth for training and evaluating.

Note: The EF source for the model was taken from the USEEIO model.

Code Architecture:

Architecture reveals a well-structured approach for:

1. Mapping business activities to appropriate emission factors.
2. It uses semantic text embeddings along with LLM and human validations.

LCA Assistant Module:

1. Module provides bridge between application and AWS Bedrock cloud-based AI models.
2. AI Model is by default Claude 3 Sonnet.
3. Generates EF recommendation with explanatory text.

```
class LCAAssistant:

    def __init__(self,
llm_model="anthropic.claude-3-sonnet-20240229-v1:0"):

        self.model_list = ["anthropic.claude-3-sonnet-20240229-v1:0"]

        self.llm_model = llm_model

        self.boto3_bedrock = get_bedrock_client()
```

Utility Function for Data Processing:

1. Handles everything from data processing to embedding generation and ground truth processing.
2. Normalises text by removing punctuation.
3. Converts all the text to lowercase.
4. Filtering out common stop words and creating representations for semantic matching.
5. SentenceTransformer generates embedding.

```
def preprocess_texts(texts):

  stop_words = spacy_stopwords.STOP_WORDS.union(set(nltk_stopwords.words("english")))

  def clean_and_tokenize(text):

    text = re.sub(r"[^\w\s]", " ", text.lower())

    return [word for word in text.split() if word not in stop_words]


def get_device():

  if torch.cuda.is_available():

    device = "cuda"

    logger.info("Using GPU to calculate semantic text embedding ...")
```

EF Database Access:

1. The system includes a specialised function for loading and processing EF databases.
2. These functions load structured data from ECOINVENT and EPA's USEEIO's model.

```
def
get_ecoinvent_data(ecoinvent_file="https://19913970.fs1.hubspotusercontent-na1.net/hubfs/19913970/Database-Overview-for-ecoinvent-v3.9.1-9.xlsx"):


def get_naics_data(

useeio_file="https://pasteur.epa.gov/uploads/10.23719/1528686/SupplyChainGHGEmissionFactors_v1.2_NAICS_CO2e_USD2021.csv",
    naics_file="https://www.census.gov/naics/2017NAICS/2017_NAICS_Index_File.xlsx"):
```

Recommendation Generation:

1. The function takes business activity description, computes its embedding and measures semantic similarity.
2. Generates a ranked list of potential matches.

```
def get_ranked_list(
    text,
    semantic_text_model,
    eco_df,
    eco_ref,
    eco_ref_embedding,
    lca_type,
):
    # Embedding generation and similarity calculation
```

```
    activity_embedding = semantic_text_model.encode([text], show_progress_bar=False,
batch_size=1)
```

Ground Truth Collection:

1. Function organises the AI recommendations into the format suitable for human review.
2. Provides an option for human annotors to select the correct emission factors.

```
def prepare_process_json(activity_text, response, sel_eco, uniq_id):
```

Technical Implementation:

1. AWS integration

```
from google.colab import userdata


# Securely access AWS credentials stored in Colab secrets

aws_access_key = userdata.get('AWS_ACCESS_KEY_ID')

aws_secret_key = userdata.get('AWS_SECRET_ACCESS_KEY')

aws_region = userdata.get('AWS_REGION', 'us-east-1')


# Set environment variables for AWS authentication

os.environ['AWS_ACCESS_KEY_ID'] = aws_access_key

os.environ['AWS_SECRET_ACCESS_KEY'] = aws_secret_key

os.environ['AWS_REGION'] = aws_region
```

2. It handles authentication, regional configurations, role assumptions, and retry logic.

Progress Tracking:

1. It ensures long-running processes provide appropriate feedback.

Identifier Management:

1. Supports various needs for deterministic hashing and random identifier generation.

References:

1. https://github.com/amazon-science/carbon-assessment-with-ml/tree/main/parakeet
2. Parakeet_Model.ipynb