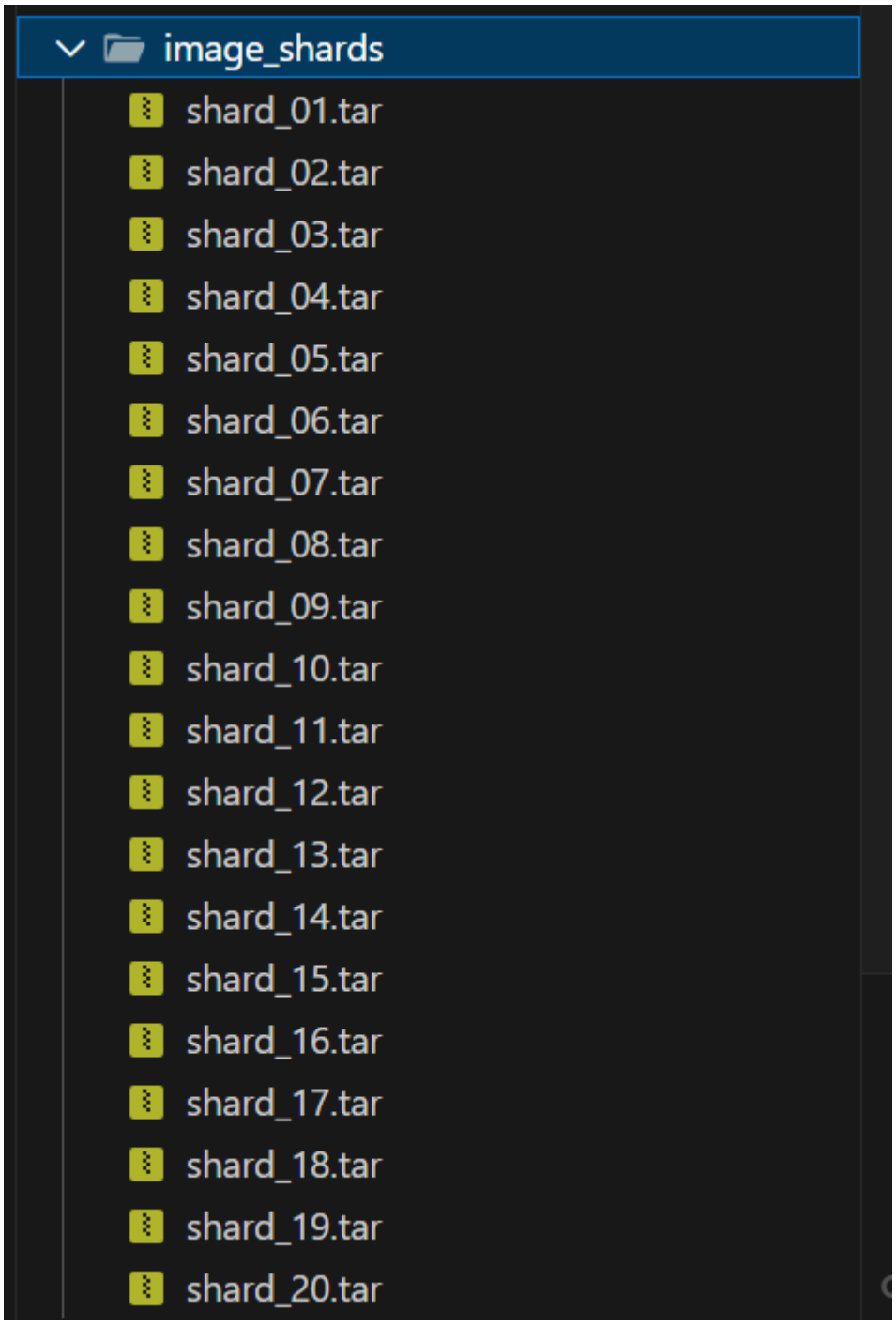
Đề tài: Xử lý dữ liệu lớn với kỹ thuật Tách nền – Nhóm 2 – 64HTTT1

Phần 1: Chạy mapreduce với Image Encoder

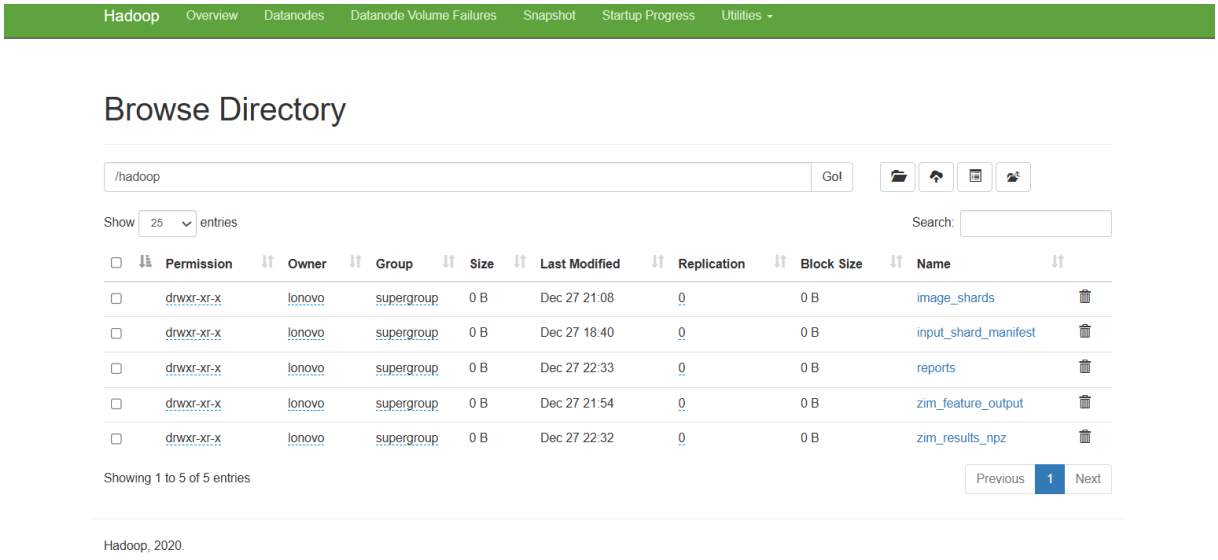
1. Chuẩn bị dữ liệu và đẩy lên HDFS

Bước 1: Chuẩn bị ảnh và nén thành các file .tar



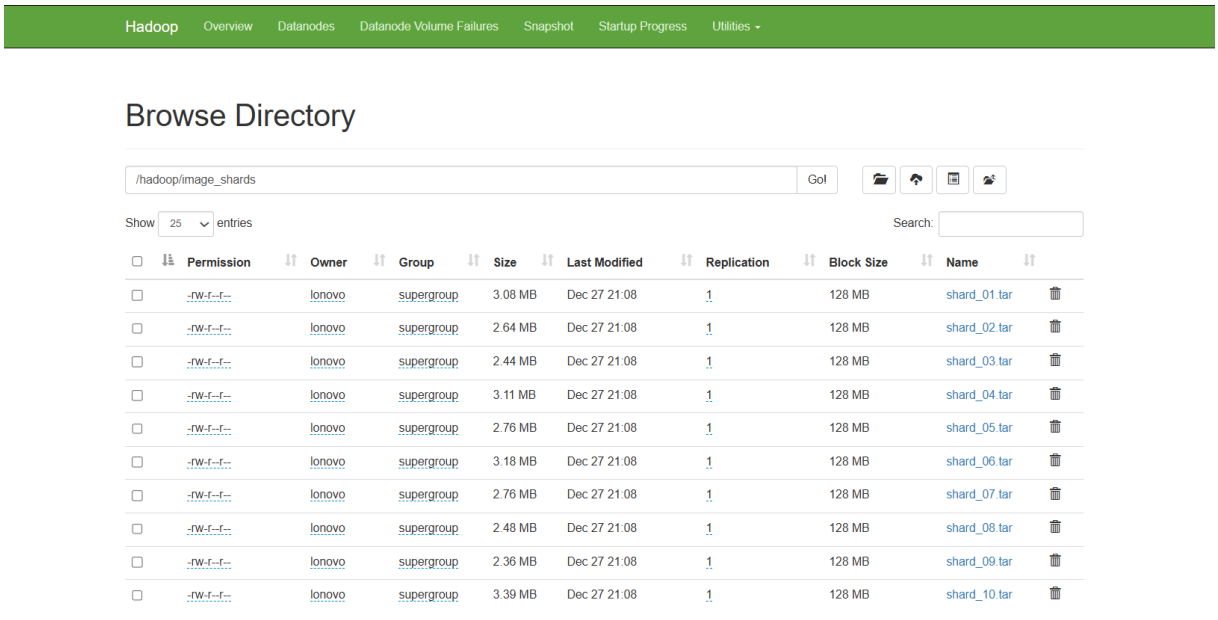
Bước 2: Tạo thư mục chứa các file .tar trên hdfs

hdfs dfs -mkdir -p /hadoop/image_shards



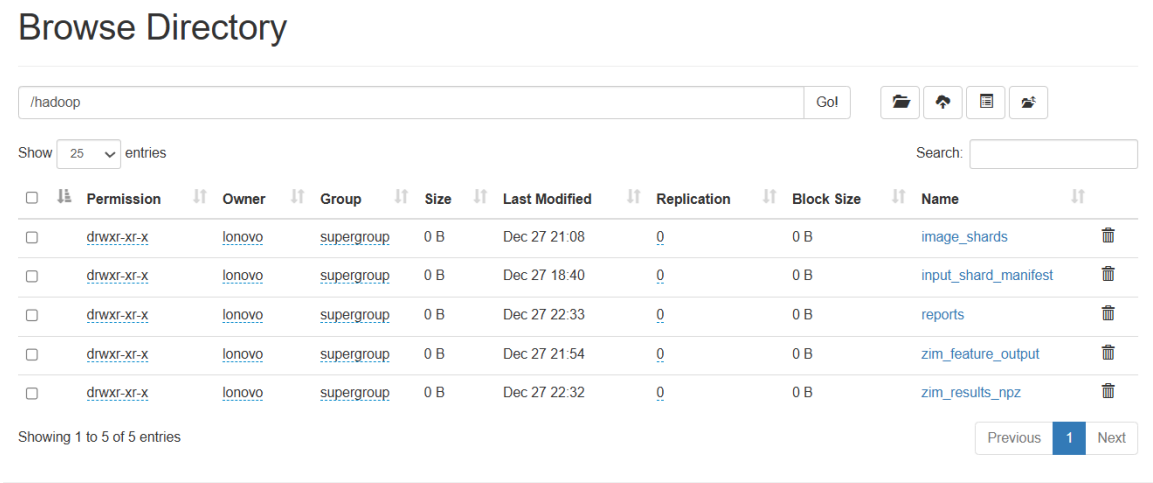
Bước 3: Đẩy file .tar lên hdfs

hdfs dfs -put "D:\Hoc_ki_7\Phan_tich_du_lieu_lon\Code\ZIM\image_shards*.tar" /hadoop/image_shards



Bước 4: Tạo thư mục chứa file đường dẫn

hdfs dfs -mkdir -p /hadoop/input_shard_manifest



Bước 5: Đẩy file chứa đường dẫn lên hdfs

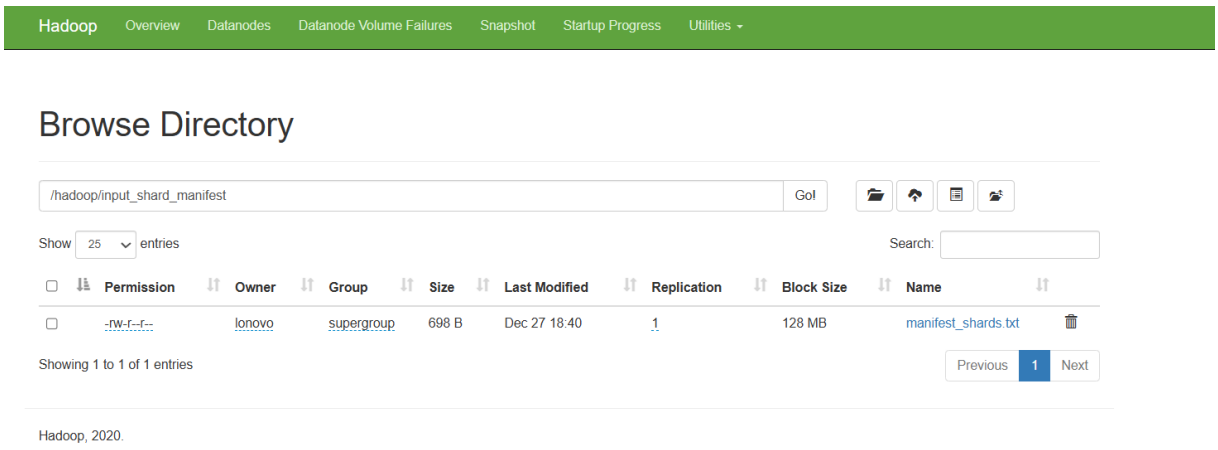
hdfs dfs -put "

"D:\Hoc_ki_7\Phan_tich_du_lieu_lon\Code\ZIM\mapreduce\manifest_shards.txt

" /hadoop/input_shard_manifest

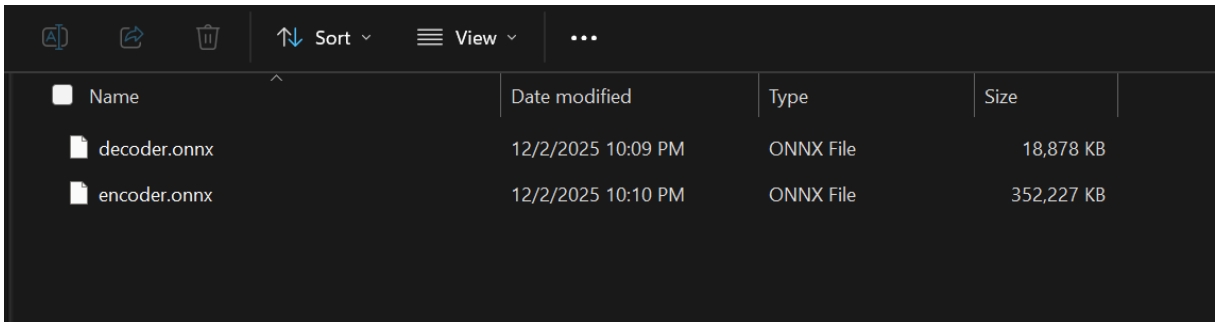
manifest_shards.txt

```
/hadoop/image_shards/shard_01.tar
/hadoop/image_shards/shard_02.tar
/hadoop/image_shards/shard_03.tar
/hadoop/image_shards/shard_04.tar
/hadoop/image_shards/shard_05.tar
/hadoop/image_shards/shard_06.tar
/hadoop/image_shards/shard_07.tar
/hadoop/image_shards/shard_08.tar
/hadoop/image_shards/shard_09.tar
/hadoop/image_shards/shard_10.tar
/hadoop/image_shards/shard_11.tar
/hadoop/image_shards/shard_12.tar
/hadoop/image_shards/shard_13.tar
/hadoop/image_shards/shard_14.tar
/hadoop/image_shards/shard_15.tar
/hadoop/image_shards/shard_16.tar
/hadoop/image_shards/shard_17.tar
/hadoop/image_shards/shard_18.tar
/hadoop/image_shards/shard_19.tar
/hadoop/image_shards/shard_20.tar
```



2. Chuẩn bị model và code để chạy mapreduce

Bước 1: Tại model encoder từ link: https://huggingface.co/naver-iv/zim-anything-vitb/tree/main/zim_vit_b_2043



Bước 2: Xây dựng code Image_Encoder có tên là zim_encoder.py

```python
#!/usr/bin/env python3
from typing import Any, Tuple

import numpy as np
import torch
import onnxruntime


def np2tensor(np_array: np.ndarray, device: torch.device) -> torch.Tensor:
    """Convert numpy array -> torch.Tensor trên device chỉ định."""
    return torch.from_numpy(np_array).to(device)


def tensor2np(torch_tensor: torch.Tensor) -> np.ndarray:
```

```
    """Convert torch.Tensor -> numpy array (luôn đưa về CPU trước)."""
    return torch_tensor.detach().cpu().numpy()


class ZIM_Encoder:
    def __init__(
        self,
        onnx_path: str,
        num_threads: int = 8,
        use_cuda: bool = False,
        device_id: int = 0,
        verbose: bool = True,
    ):
        """
        onnx_path: đường dẫn tới file .onnx của image encoder ZIM
        num_threads: số thread cho onnxruntime
        use_cuda: nếu True sẽ cố gắng dùng CUDAExecutionProvider
        device_id: GPU id (nếu dùng CUDA)
        verbose: in thông tin input/output của model
        """
        self.onnx_path = onnx_path

        session_options = onnxruntime.SessionOptions()
        session_options.intra_op_num_threads = num_threads
        session_options.inter_op_num_threads = num_threads

        # Mặc định luôn có CPU provider
        providers = ["CPUExecutionProvider"]
        provider_options = []

        if use_cuda:
            # Thử thêm CUDA provider (nếu đã cài onnxruntime-gpu + có CUDA)
            try:
                providers.insert(0, "CUDAExecutionProvider")
                provider_options.insert(0, {"device_id": device_id})
                if verbose:
                    print(f"[ZIM_Encoder] Trying CUDAExecutionProvider on device
{device_id}")
            except Exception as e:
```

```
                if verbose:
                    print(f"[ZIM_Encoder] Cannot enable CUDAExecutionProvider,
fallback to CPU. Reason: {e}")

        # Khởi tạo session
        self.ort_session = onnxruntime.InferenceSession(
            onnx_path,
            sess_options=session_options,
            providers=providers,
            provider_options=provider_options if provider_options else None,
        )

        # Lưu lại tên input chính (lấy input đầu tiên)
        inputs = self.ort_session.get_inputs()
        if len(inputs) == 0:
            raise RuntimeError("ONNX model has no inputs.")
        self.input_name = inputs[0].name

        if verbose:
            print(f"[ZIM_Encoder] Using input name: {self.input_name}")
            print("[ZIM_Encoder] Model outputs:")
            for i, out in enumerate(self.ort_session.get_outputs()):
                print(f"  #{i}: name={out.name}, shape={out.shape},
type={out.type}")

    def forward(
        self,
        image: torch.Tensor,
    ) -> Tuple[torch.Tensor, Tuple[torch.Tensor, torch.Tensor, torch.Tensor]]:
        """
        image: torch.Tensor shape (B, 3, 1024, 1024) trên CPU hoặc GPU
        returns: (image_embeddings, (feat_D0, feat_D1, feat_D2))

        Lưu ý:
        - Hàm này giả định model ONNX trả về 4 output:
          [0] image_embeddings
          [1] feat_D0
          [2] feat_D1
```

```
          [3] feat_D2
        """
        if not isinstance(image, torch.Tensor):
            raise TypeError("image must be a torch.Tensor")

        if image.ndim != 4 or image.shape[1] != 3 or image.shape[2] != 1024 or
image.shape[3] != 1024:
            raise ValueError(
                f"Expected image shape (B, 3, 1024, 1024) but got
{tuple(image.shape)}. "
                "Hãy chắc chắn đã resize/normalize đúng trước khi gọi encoder."
            )

        device = image.device

        # Chuẩn bị numpy input (luôn về CPU)
        image_np = tensor2np(image)
        ort_inputs = {self.input_name: image_np}

        # Chạy ONNX inference
        outputs = self.ort_session.run(None, ort_inputs)

        if len(outputs) != 4:
            raise RuntimeError(
                f"Expected 4 outputs (image_embeddings, feat_D0, feat_D1, feat_D2)
but got {len(outputs)}. "
                "Kiểm tra lại model ONNX hoặc chỉnh lại wrapper cho đúng số
output."
            )

        image_embeddings_np, feat_D0_np, feat_D1_np, feat_D2_np = outputs

        # Convert ngược về torch.Tensor trên device ban đầu
        image_embeddings = np2tensor(image_embeddings_np, device)
        feat_D0 = np2tensor(feat_D0_np, device)
        feat_D1 = np2tensor(feat_D1_np, device)
        feat_D2 = np2tensor(feat_D2_np, device)
```

```
        return image_embeddings, (feat_D0, feat_D1, feat_D2)

    def __call__(self, *args, **kwargs) -> Any:
        return self.forward(*args, **kwargs)
```

Bước 3: Tạo file mapper.py cùng thư mục với model và zim_encoder.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import os
import sys
import io
import tarfile
import shutil
import tempfile
import subprocess
import traceback
import gc
import numpy as np
import torch
from PIL import Image
from zim_encoder import ZIM_Encoder

# ================== CẤU HÌNH MÔI TRƯỜNG ==================
extra_paths = [
    r"C:\Users\lonovo\AppData\Local\Programs\Python\Python311\Lib\site-packages",
    r"C:\Users\lonovo\AppData\Roaming\Python\Python311\site-packages",
    r"C:\PythonLibs"
]
for p in extra_paths:
    if os.path.exists(p) and p not in sys.path:
        sys.path.append(p)

HADOOP_CMD = os.path.join(os.environ.get("HADOOP_HOME", r"C:\hadoop-3.3.0"),
"bin", "hadoop.cmd")
HDFS_TAR_DIR = "/hadoop/image_shards"
HDFS_NPZ_OUT_DIR = "/hadoop/zim_results_npz"
```

```
ONNX_FILE = "zim_model.onnx"
BATCH_SIZE = 4 # Tăng tốc độ bằng cách xử lý nhiều ảnh cùng lúc

def hadoop_fs(*args):
    return subprocess.check_call([HADOOP_CMD, "fs", *args], shell=True)

def preprocess_image(img_bytes):
    """Tiền xử lý ảnh đơn lẻ trả về numpy array"""
    img = Image.open(io.BytesIO(img_bytes)).convert("RGB")
    if img.size != (1024, 1024):
        img = img.resize((1024, 1024))
    return np.transpose(np.asarray(img, dtype=np.float32) / 255.0, (2, 0, 1))

def main():
    if not os.path.exists(ONNX_FILE):
        sys.stderr.write(f"FATAL: Khong tim thay model {ONNX_FILE}\n")
        sys.exit(1)

    # Khởi tạo Encoder một lần duy nhất
    encoder = ZIM_Encoder(onnx_path=ONNX_FILE, use_cuda=torch.cuda.is_available(),
verbose=False)
    workdir = tempfile.mkdtemp(prefix="zim_batch_map_")

    try:
        for raw in sys.stdin:
            line = raw.strip()
            if not line: continue

            hdfs_tar = line if line.startswith("/") else
f"{HDFS_TAR_DIR}/{os.path.basename(line)}"
            shard_id = os.path.splitext(os.path.basename(hdfs_tar))[0]
            local_tar = os.path.join(workdir, f"{shard_id}.tar")
            results_dict = {}

            try:
                hadoop_fs("-get", "-f", hdfs_tar, local_tar)

                with tarfile.open(local_tar, "r:*") as tf:
```

```
                    batch_images = []
                    batch_names = []

                    for m in tf:
                        if not m.isfile() or not m.name.lower().endswith(('.jpg',
'.png', '.jpeg')):
                            continue

                        f = tf.extractfile(m)
                        if not f: continue

                        try:
                            # 1. Thu thập ảnh vào batch
                            batch_images.append(preprocess_image(f.read()))
                            batch_names.append(m.name)

                            # 2. Xử lý khi batch đầy
                            if len(batch_images) == BATCH_SIZE:
                                input_tensor =
torch.from_numpy(np.stack(batch_images))
                                emb, _ = encoder(input_tensor)

                                # Lưu kết quả
                                emb_np = emb.detach().cpu().numpy()
                                for i, name in enumerate(batch_names):
                                    results_dict[name] = emb_np[i]

                                # Giải phóng bộ nhớ ngay lập tức
                                del input_tensor, emb, emb_np
                                batch_images, batch_names = [], []
                                if torch.cuda.is_available():
torch.cuda.empty_cache()

                        except Exception as img_err:
                            sys.stderr.write(f"Warning: Loi anh {m.name}:
{str(img_err)}\n")

                    # 3. Xử lý nốt các ảnh dư thừa cuối batch
```

```
                    if batch_images:
                        input_tensor = torch.from_numpy(np.stack(batch_images))
                        emb, _ = encoder(input_tensor)
                        emb_np = emb.detach().cpu().numpy()
                        for i, name in enumerate(batch_names):
                            results_dict[name] = emb_np[i]
                        del input_tensor, emb, emb_np

                # BƯỚC 3: Lưu và Upload NPZ
                if results_dict:
                    local_npz = os.path.join(workdir, f"{shard_id}.npz")
                    np.savez_compressed(local_npz, **results_dict)

                    hdfs_npz_path = f"{HDFS_NPZ_OUT_DIR}/{shard_id}.npz"
                    hadoop_fs("-mkdir", "-p", HDFS_NPZ_OUT_DIR)
                    hadoop_fs("-put", "-f", local_npz, hdfs_npz_path)
                    print(f"NPZ_PATH\t{hdfs_npz_path}")

            except Exception:
                sys.stderr.write(f"ERROR: Shard {shard_id} that bai:
{traceback.format_exc()}\n")
            finally:
                # Dọn dẹp mạnh mẽ sau mỗi Shard
                if os.path.exists(local_tar): os.remove(local_tar)
                results_dict.clear()
                gc.collect()

    finally:
        shutil.rmtree(workdir, ignore_errors=True)

if __name__ == "__main__":
    main()
```

Bước 4: Tạo file reducer.py cùng thư mục với mapper.py và zim_encoder.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
```

```python
import sys
import os
import io # Đã bổ sung import này
import subprocess
import numpy as np
import tempfile
import shutil
from datetime import datetime

# Ép kiểu stdout và stderr sử dụng UTF-8 để in được tiếng Việt trên Windows/Hadoop
sys.stdout = io.TextIOWrapper(sys.stdout.buffer, encoding='utf-8')
sys.stderr = io.TextIOWrapper(sys.stderr.buffer, encoding='utf-8')

# ================== CẤU HÌNH HỆ THỐNG ==================
HADOOP_HOME = os.environ.get("HADOOP_HOME", r"C:\hadoop-3.3.0")
HADOOP_CMD = os.path.join(HADOOP_HOME, "bin", "hadoop.cmd")
HDFS_REPORT_DIR = "/hadoop/reports/zim_evaluation"

def hadoop_fs(*args):
    """Thực thi lệnh shell HDFS"""
    cmd = [HADOOP_CMD, "fs", *args]
    try:
        subprocess.check_call(cmd, shell=True)
    except subprocess.CalledProcessError as e:
        sys.stderr.write(f"Reducer HDFS Error: {e}\n")

def calculate_cosine_similarity(vec_a, vec_b):
    """
    Tính độ tương đồng Cosine giữa hai vector:
    $$\text{Similarity} = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|
\|\mathbf{B}\|}$$
    """
    dot_product = np.dot(vec_a, vec_b)
    norm_a = np.linalg.norm(vec_a)
    norm_b = np.linalg.norm(vec_b)
    return dot_product / (norm_a * norm_b + 1e-8)

def main():
```

```
    all_embeddings = []
    workdir = tempfile.mkdtemp(prefix="zim_red_final_")

    try:
        # BƯỚC 1: Thu thập toàn bộ dữ liệu từ các file .npz (Mapper gửi sang)
        for line in sys.stdin:
            line = line.strip()
            if not line: continue

            try:
                # Phân tách Key-Value từ Shuffle & Sort (NPZ_PATH \t path)
                parts = line.split("\t")
                if len(parts) < 2: continue
                hdfs_npz_path = parts[1]

                local_npz = os.path.join(workdir, os.path.basename(hdfs_npz_path))
                hadoop_fs("-get", "-f", hdfs_npz_path, local_npz)

                with np.load(local_npz) as data:
                    for img_name in data.files:
                        all_embeddings.append((img_name,
data[img_name].flatten()))

                if os.path.exists(local_npz): os.remove(local_npz)
            except Exception as e:
                sys.stderr.write(f"Lỗi khi xử lý file NPZ: {str(e)}\n")

        if not all_embeddings:
            sys.stderr.write("Không tìm thấy dữ liệu để xử lý.\n")
            return

        # BƯỚC 2: Phân tích thống kê & Tính toán Global Centroid
        feature_matrix = np.array([item[1] for item in all_embeddings])
        global_centroid = np.mean(feature_matrix, axis=0)

        total_mean = np.mean(feature_matrix)
        total_std = np.std(feature_matrix)
        threshold = 0.7  # Ngưỡng tin cậy đánh giá đặc trưng ViT-B
```

```
        # BƯỚC 3: Xây dựng nội dung báo cáo
        report = []
        report.append("="*80)
        report.append(f"HỆ THỐNG ĐÁNH GIÁ ĐỘ TIN CẬY IMAGE ENCODER (ViT-B)")
        report.append(f"Thời gian tạo báo cáo:
{datetime.now().strftime('%Y-%m-%d %H:%M:%S')}")
        report.append("="*80)
        report.append(f"{'IMAGE NAME':<50} | {'RELIABILITY SCORE':<20}")
        report.append("-"*80)

        reliable_count = 0
        for img_name, emb_vector in all_embeddings:
            score = calculate_cosine_similarity(emb_vector, global_centroid)
            status = "OK" if score >= threshold else "LOW_CONFIDENCE"
            if score >= threshold: reliable_count += 1
            report.append(f"{img_name:<50} | {score:.4f} [{status}]")

        summary = [
            "-"*80,
            "TỔNG KẾT THỐNG KÊ:",
            f"- Tổng số lượng ảnh: {len(all_embeddings)}",
            f"- Giá trị trung bình Embedding (Mean): {total_mean:.6f}",
            f"- Độ lệch chuẩn Embedding (Std): {total_std:.6f}",
            f"- Số lượng ảnh đạt chuẩn (>{threshold}): {reliable_count}",
            f"- Tỷ lệ tin cậy toàn hệ thống:
{(reliable_count/len(all_embeddings))*100:.2f}%",
            "="*80
        ]
        report.extend(summary)

        # BƯỚC 4: Lưu báo cáo vào HDFS
        final_report_str = "\n".join(report)
        local_report_file = os.path.join(workdir, "zim_reliability_report.txt")

        with open(local_report_file, "w", encoding="utf-8") as f:
            f.write(final_report_str)
```

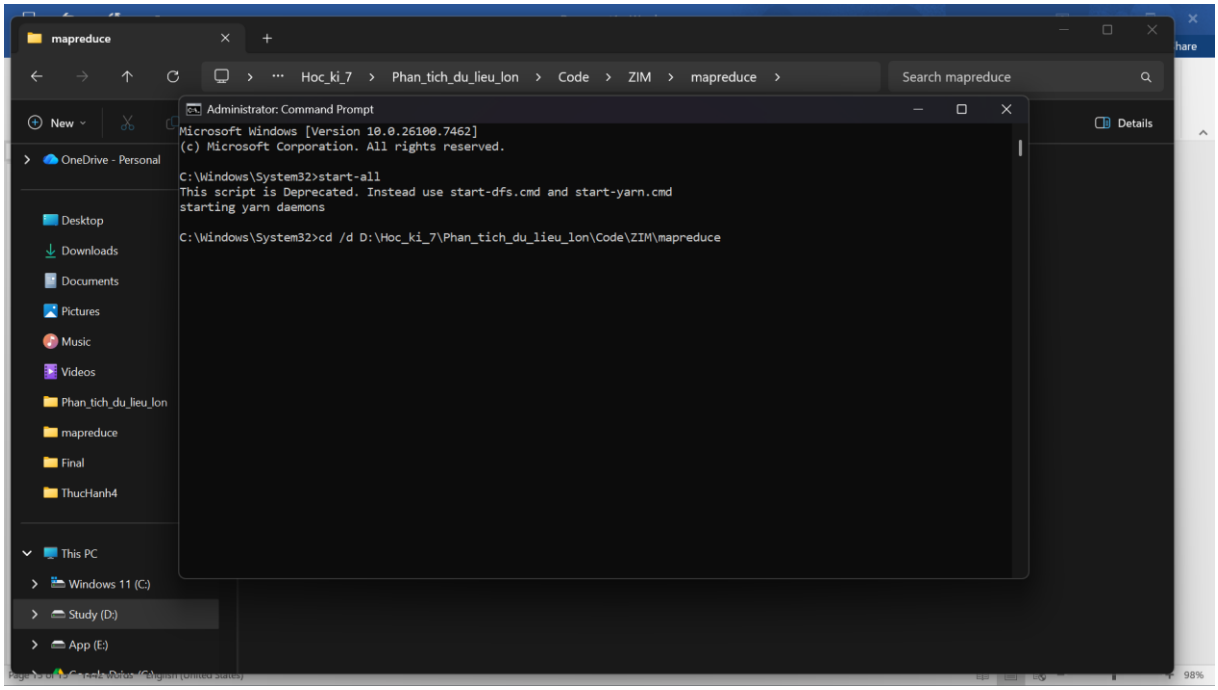
```
        hadoop_fs("-mkdir", "-p", HDFS_REPORT_DIR)
        hadoop_fs("-put", "-f", local_report_file,
f"{HDFS_REPORT_DIR}/reliability_report.txt")

        # BƯỚC 5: Xuất ra stdout để ghi vào thư mục output của Job
        print(final_report_str)

    finally:
        shutil.rmtree(workdir, ignore_errors=True)

if __name__ == "__main__":
    main()
```

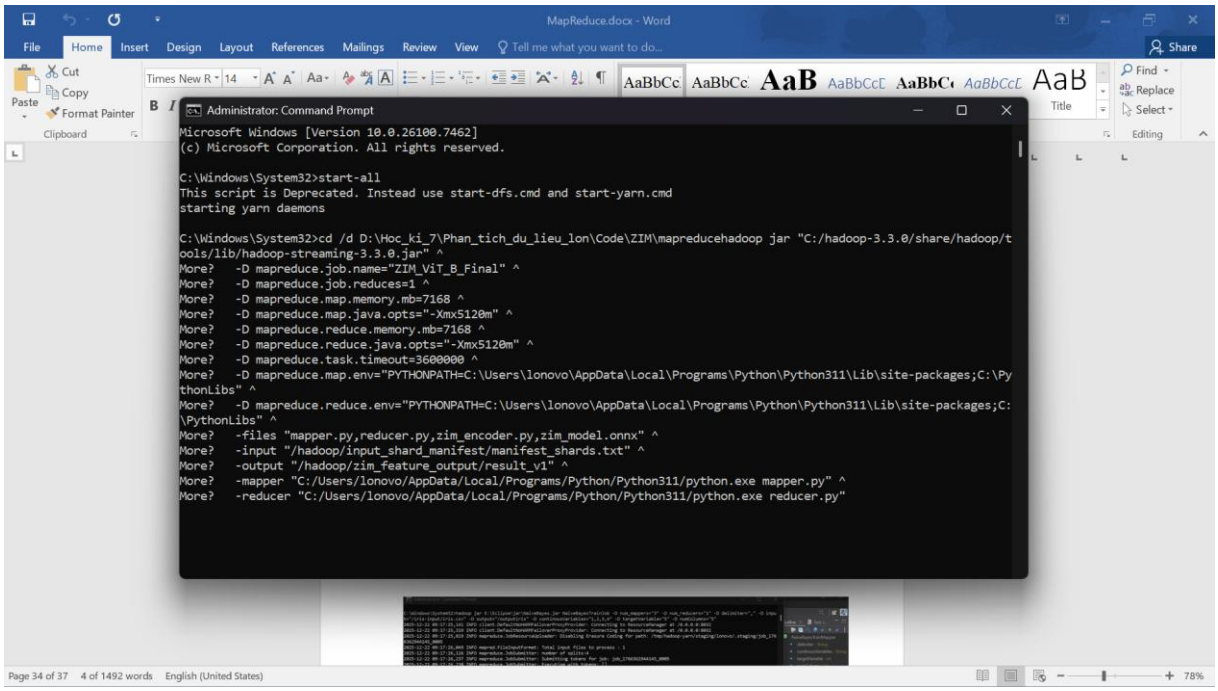
Bước 5: Di chuyển đến thư mục chứa các file code



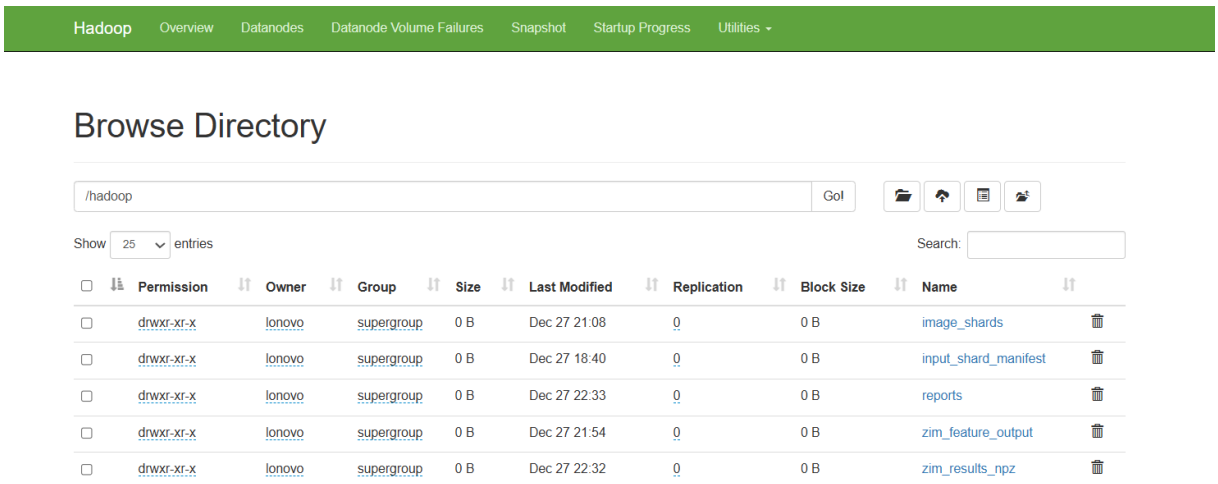
Bước 6: Chạy chương trình

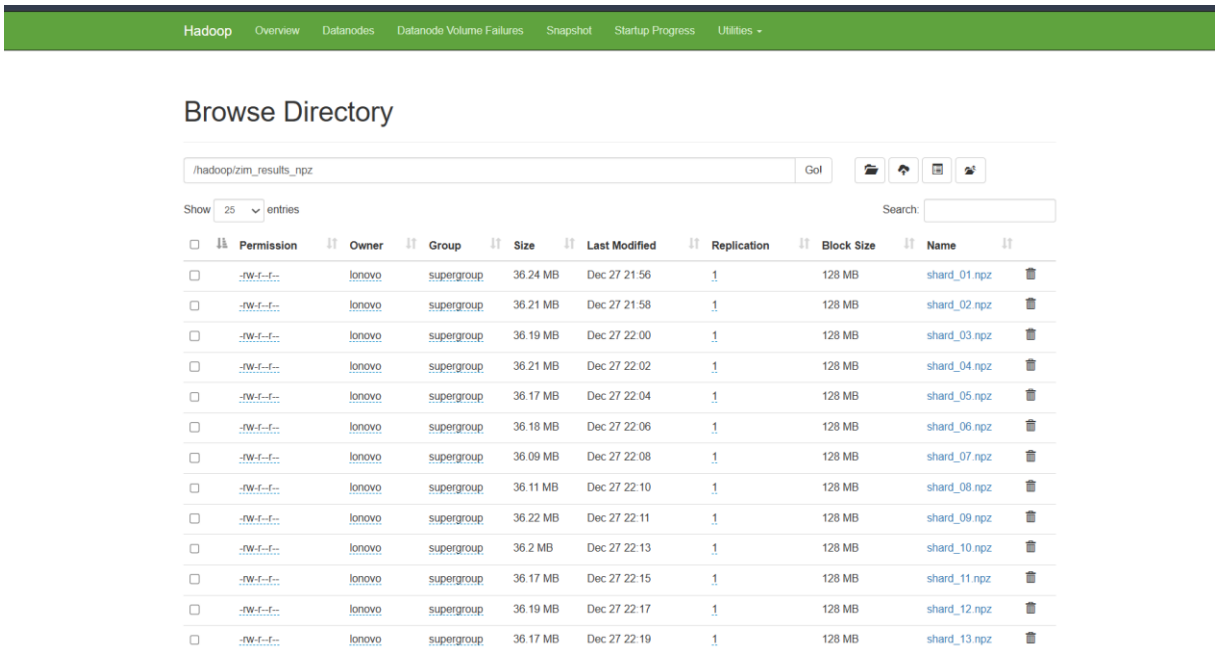
```
hadoop jar "C:/hadoop-3.3.0/share/hadoop/tools/lib/hadoop-streaming-3.3.0.jar" ^
  -D mapreduce.job.name="ZIM_ViT_B_Final" ^
  -D mapreduce.job.reduces=1 ^
  -D mapreduce.map.memory.mb=7168 ^
  -D mapreduce.map.java.opts="-Xmx5120m" ^
```

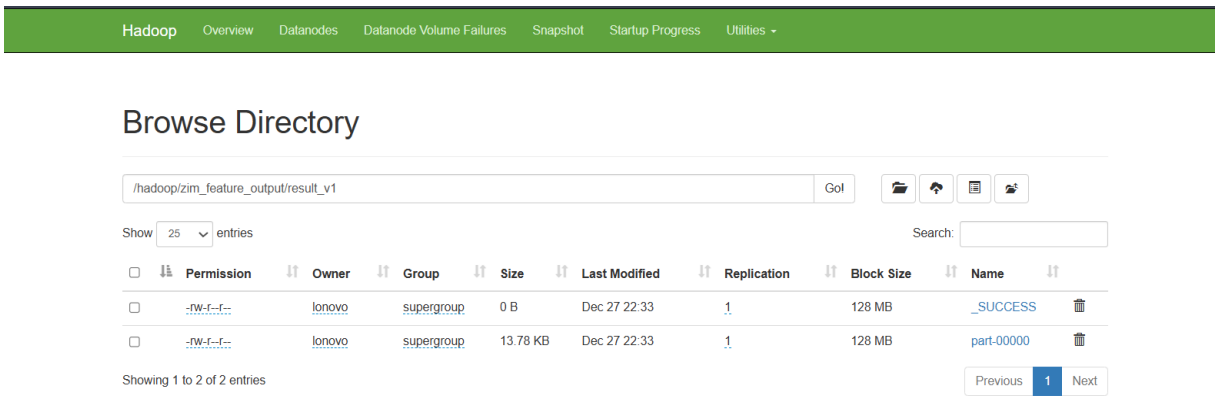
```
  -D mapreduce.reduce.memory.mb=7168 ^
  -D mapreduce.reduce.java.opts="-Xmx5120m" ^
  -D mapreduce.task.timeout=3600000 ^
  -D
mapreduce.map.env="PYTHONPATH=C:\Users\lonovo\AppData\Local\Programs\Python\Python311\Lib\site-packages;C:\PythonLibs" ^
  -D
mapreduce.reduce.env="PYTHONPATH=C:\Users\lonovo\AppData\Local\Programs\Python\Python311\Lib\site-packages;C:\PythonLibs" ^
  -files "mapper.py,reducer.py,zim_encoder.py,zim_model.onnx" ^
  -input "/hadoop/input_shard_manifest/manifest_shards.txt" ^
  -output "/hadoop/zim_feature_output/result_v1" ^
  -mapper
"C:/Users/lonovo/AppData/Local/Programs/Python/Python311/python.exe
mapper.py" ^
  -reducer
"C:/Users/lonovo/AppData/Local/Programs/Python/Python311/python.exe
reducer.py"
```

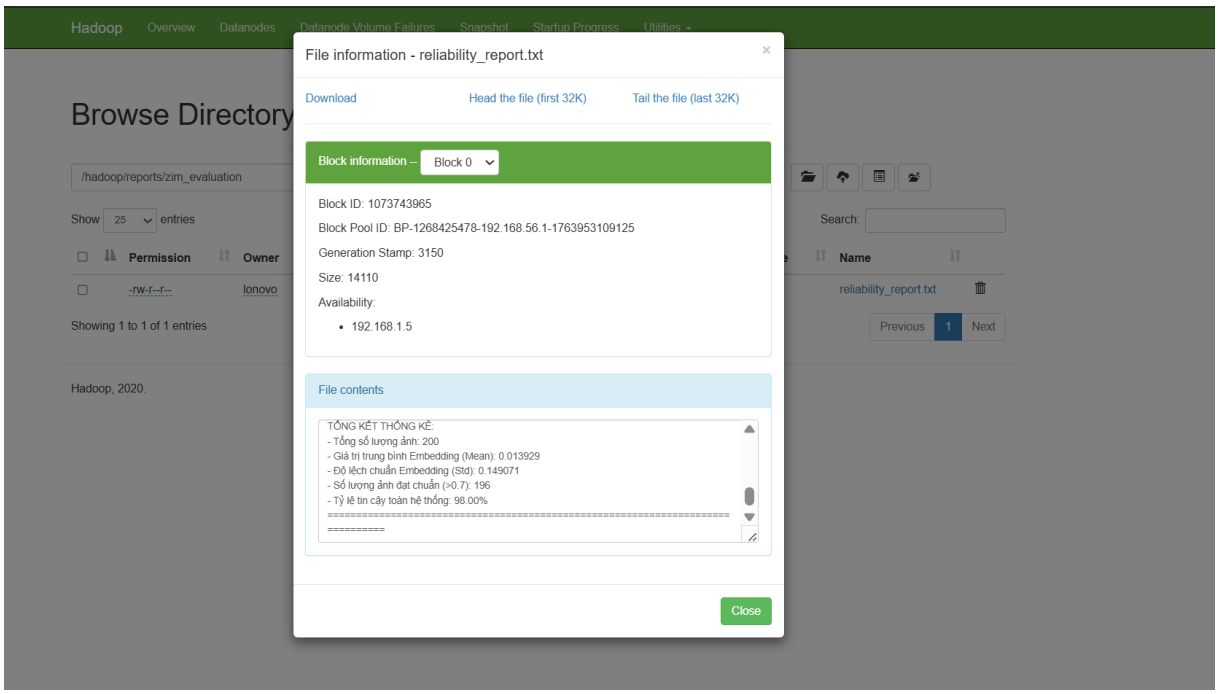


Bước 7: Kiểm tra kết quả thu được


Hadoop
Overview
Datanodes
Datanode Volume Failures
Snapshot
Startup Progress
Utilities
Browse Directory
/hadoop
Go!
Show 25 entries
Search:
Permission
Owner
Group
Size
Last Modified
Replication
Block Size
Name
drwxr-xr-x lonovo supergroup 0 B Dec 27 21:08 0 0 B image_shards
drwxr-xr-x lonovo supergroup 0 B Dec 27 18:40 0 0 B input_shard_manifest
drwxr-xr-x lonovo supergroup 0 B Dec 27 22:33 0 0 B reports
drwxr-xr-x lonovo supergroup 0 B Dec 27 21:54 0 0 B zim_feature_output
drwxr-xr-x lonovo supergroup 0 B Dec 27 22:32 0 0 B zim_results_npz
Showing 1 to 5 of 5 entries
Previous
1
Next


Hadoop
Overview
Datanodes
Datanode Volume Failures
Snapshot
Startup Progress
Utilities
Browse Directory
/hadoop/zim_results_npz
Go!
Show 25 entries
Search:
Permission
Owner
Group
Size
Last Modified
Replication
Block Size
Name
-rw-r--r-- lonovo supergroup 36.24 MB Dec 27 21:56 1 128 MB shard_01.npz
-rw-r--r-- lonovo supergroup 36.21 MB Dec 27 21:58 1 128 MB shard_02.npz
-rw-r--r-- lonovo supergroup 36.19 MB Dec 27 22:00 1 128 MB shard_03.npz
-rw-r--r-- lonovo supergroup 36.21 MB Dec 27 22:02 1 128 MB shard_04.npz
-rw-r--r-- lonovo supergroup 36.17 MB Dec 27 22:04 1 128 MB shard_05.npz
-rw-r--r-- lonovo supergroup 36.18 MB Dec 27 22:06 1 128 MB shard_06.npz
-rw-r--r-- lonovo supergroup 36.09 MB Dec 27 22:08 1 128 MB shard_07.npz
-rw-r--r-- lonovo supergroup 36.11 MB Dec 27 22:10 1 128 MB shard_08.npz
-rw-r--r-- lonovo supergroup 36.22 MB Dec 27 22:11 1 128 MB shard_09.npz
-rw-r--r-- lonovo supergroup 36.2 MB Dec 27 22:13 1 128 MB shard_10.npz
-rw-r--r-- lonovo supergroup 36.17 MB Dec 27 22:15 1 128 MB shard_11.npz
-rw-r--r-- lonovo supergroup 36.19 MB Dec 27 22:17 1 128 MB shard_12.npz
-rw-r--r-- lonovo supergroup 36.17 MB Dec 27 22:19 1 128 MB shard_13.npz


Hadoop
Overview
Datanodes
Datanode Volume Failures
Snapshot
Startup Progress
Utilities
Browse Directory
/hadoop/zim_feature_output/result_v1
Go!
Show 25 entries
Search:
Permission
Owner
Group
Size
Last Modified
Replication
Block Size
Name
-rw-r--r-- lonovo supergroup 0 B Dec 27 22:33 1 128 MB _SUCCESS
-rw-r--r-- lonovo supergroup 13.78 KB Dec 27 22:33 1 128 MB part-00000
Showing 1 to 2 of 2 entries
Previous
1
Next


Hadoop
Overview
Datanodes
Datanode Volume Failures
Snapshot
Startup Progress
Utilities
Browse Directory
/hadoop/reports/zim_evaluation
Show 25 entries
Search:
Permission
Owner
Name
-rw-r--r--
lonovo
reliability_report.txt
Showing 1 to 1 of 1 entries
Previous
1
Next
Hadoop, 2020.
File information - reliability_report.txt
Download
Head the file (first 32K)
Tail the file (last 32K)
Block information --
Block 0
Block ID: 1073743965
Block Pool ID: BP-1268425478-192.168.56.1-1763953109125
Generation Stamp: 3150
Size: 14110
Availability:
192.168.1.5
File contents
TỔNG KẾT THỐNG KÊ:
- Tổng số lượng ảnh: 200
- Giá trị trung bình Embedding (Mean): 0.013929
- Độ lệch chuẩn Embedding (Std): 0.149071
- Số lượng ảnh đạt chuẩn (>0.7): 196
- Tỷ lệ tin cậy toàn hệ thống: 98.00%
==========
Close

Phần 2: Chạy Map-Reduce với Prompt Encoder + Transformer Decoder

Bước 1: Chuẩn bị dữ liệu

1. Dữ liệu dạng RBG Image (.jpg)

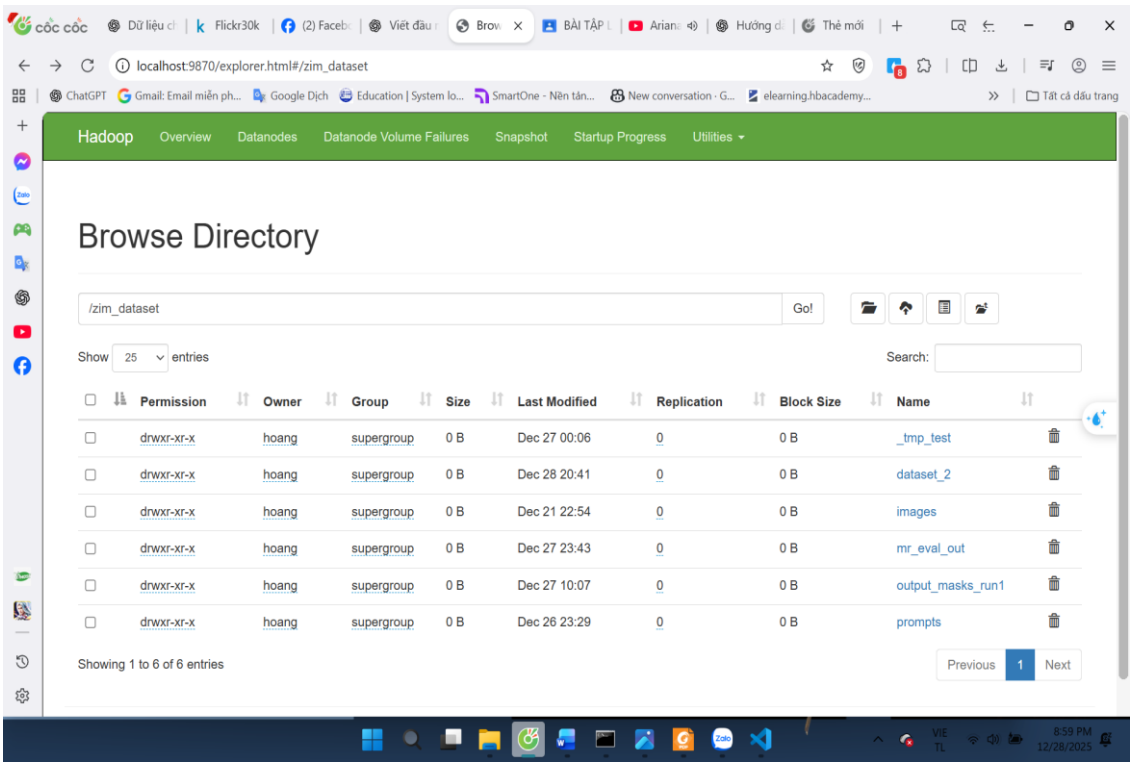
2. Dữ liệu của các Point, Box

```
"points": [[810, 540]], "point_labels": [1], "box": [405, 270, 1215, 810], "image": "hdfs:///zim_dat
"points": [[540, 810]], "point_labels": [1], "box": [270, 405, 810, 1215], "image": "hdfs:///zim_dat
"points": [[540, 570]], "point_labels": [1], "box": [270, 285, 810, 855], "image": "hdfs:///zim_data
"points": [[540, 810]], "point_labels": [1], "box": [270, 405, 810, 1215], "image": "hdfs:///zim_dat
"points": [[815, 540]], "point_labels": [1], "box": [408, 270, 1223, 810], "image": "hdfs:///zim_dat
"points": [[540, 810]], "point_labels": [1], "box": [270, 405, 810, 1215], "image": "hdfs:///zim_dat
"points": [[814, 540]], "point_labels": [1], "box": [407, 270, 1222, 810], "image": "hdfs:///zim_dat
"points": [[540, 810]], "point_labels": [1], "box": [270, 405, 810, 1215], "image": "hdfs:///zim_dat
"points": [[959, 540]], "point_labels": [1], "box": [480, 270, 1439, 810], "image": "hdfs:///zim_dat
"points": [[810, 540]], "point_labels": [1], "box": [405, 270, 1215, 810], "image": "hdfs:///zim_dat
"points": [[808, 540]], "point_labels": [1], "box": [404, 270, 1212, 810], "image": "hdfs:///zim_dat
"points": [[809, 540]], "point_labels": [1], "box": [405, 270, 1214, 810], "image": "hdfs:///zim_dat
"points": [[810, 540]], "point_labels": [1], "box": [405, 270, 1215, 810], "image": "hdfs:///zim_dat
"points": [[810, 540]], "point_labels": [1], "box": [405, 270, 1215, 810], "image": "hdfs:///zim_dat
"points": [[540, 810]], "point_labels": [1], "box": [270, 405, 810, 1215], "image": "hdfs:///zim_dat
"points": [[870, 540]], "point_labels": [1], "box": [435, 270, 1306, 810], "image": "hdfs:///zim_dat
"points": [[540, 809]], "point_labels": [1], "box": [270, 405, 810, 1214], "image": "hdfs:///zim_dat
"points": [[719, 540]], "point_labels": [1], "box": [360, 270, 1079, 810], "image": "hdfs:///zim_dat
"points": [[810, 540]], "point_labels": [1], "box": [405, 270, 1215, 810], "image": "hdfs:///zim_dat
"points": [[809, 540]], "point_labels": [1], "box": [405, 270, 1214, 810], "image": "hdfs:///zim_dat
"points": [[810, 540]], "point_labels": [1], "box": [405, 270, 1215, 810], "image": "hdfs:///zim_dat
"points": [[810, 540]], "point_labels": [1], "box": [405, 270, 1215, 810], "image": "hdfs:///zim_dat
"points": [[540, 810]], "point_labels": [1], "box": [270, 405, 810, 1215], "image": "hdfs:///zim_dat
"points": [[959, 540]], "point_labels": [1], "box": [480, 270, 1439, 810], "image": "hdfs:///zim_dat
"points": [[810, 540]], "point_labels": [1], "box": [405, 270, 1215, 810], "image": "hdfs:///zim_dat
"points": [[540, 726]], "point_labels": [1], "box": [270, 363, 810, 1090], "image": "hdfs:///zim_dat
"points": [[810, 540]], "point_labels": [1], "box": [405, 270, 1215, 810], "image": "hdfs:///zim_dat
"points": [[810, 540]], "point_labels": [1], "box": [405, 270, 1215, 810], "image": "hdfs:///zim_dat
"points": [[810, 540]], "point_labels": [1], "box": [405, 270, 1215, 810], "image": "hdfs:///zim_dat
```


Bước 2: Tạo thư mục chứa các file dữ liệu trên hdfs

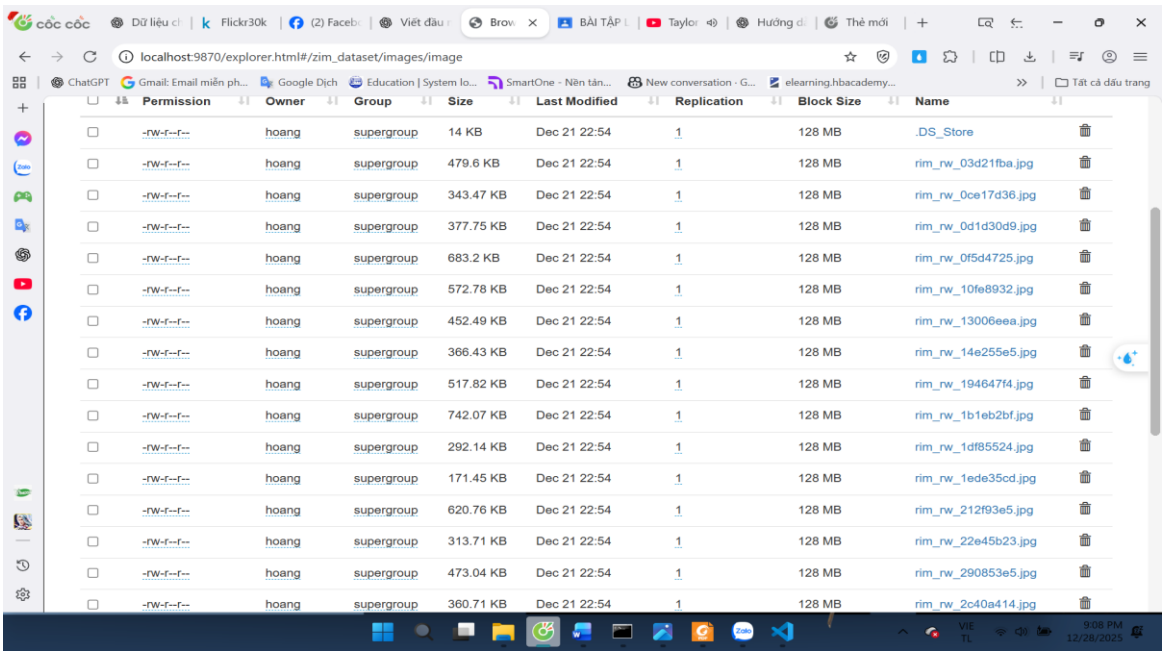
hdfs dfs -mkdir -p /zim_dataset/images

hdfs dfs -mkdir -p /zim_dataset/ prompts



Bước 3: Đẩy dữ liệu gồm các ảnh và các Prompt chứa tọa độ các Point, box lên HDFS

hdfs dfs -put C:\Users\hoang\Downloads\TestMapreduce\image* /zim_dataset/images/



Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rw-r--r--	hoang	supergroup	14 KB	Dec 21 22:54	1	128 MB	.DS_Store
-rw-r--r--	hoang	supergroup	479.6 KB	Dec 21 22:54	1	128 MB	rim_rw_03d21fba.jpg
-rw-r--r--	hoang	supergroup	343.47 KB	Dec 21 22:54	1	128 MB	rim_rw_0ce17d36.jpg
-rw-r--r--	hoang	supergroup	377.75 KB	Dec 21 22:54	1	128 MB	rim_rw_0d1d30d9.jpg
-rw-r--r--	hoang	supergroup	683.2 KB	Dec 21 22:54	1	128 MB	rim_rw_0f5d4725.jpg
-rw-r--r--	hoang	supergroup	572.78 KB	Dec 21 22:54	1	128 MB	rim_rw_10fe8932.jpg
-rw-r--r--	hoang	supergroup	452.49 KB	Dec 21 22:54	1	128 MB	rim_rw_13006eea.jpg
-rw-r--r--	hoang	supergroup	366.43 KB	Dec 21 22:54	1	128 MB	rim_rw_14e255e5.jpg
-rw-r--r--	hoang	supergroup	517.82 KB	Dec 21 22:54	1	128 MB	rim_rw_194647f4.jpg
-rw-r--r--	hoang	supergroup	742.07 KB	Dec 21 22:54	1	128 MB	rim_rw_1b1eb2bf.jpg
-rw-r--r--	hoang	supergroup	292.14 KB	Dec 21 22:54	1	128 MB	rim_rw_1df85524.jpg
-rw-r--r--	hoang	supergroup	171.45 KB	Dec 21 22:54	1	128 MB	rim_rw_1ede35cd.jpg
-rw-r--r--	hoang	supergroup	620.76 KB	Dec 21 22:54	1	128 MB	rim_rw_212f93e5.jpg
-rw-r--r--	hoang	supergroup	313.71 KB	Dec 21 22:54	1	128 MB	rim_rw_22e45b23.jpg
-rw-r--r--	hoang	supergroup	473.04 KB	Dec 21 22:54	1	128 MB	rim_rw_290853e5.jpg
-rw-r--r--	hoang	supergroup	360.71 KB	Dec 21 22:54	1	128 MB	rim_rw_2c40a414.jpg

hdfs dfs -put prompts_hdfs.jsonl /zim_dataset/prompts/

Browse Directory

/zim_dataset/prompts Go!

Show 25 entries Search:

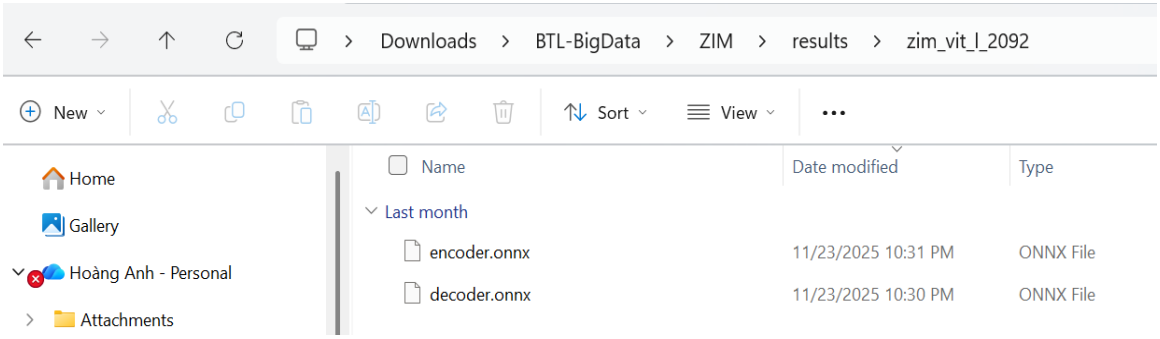
Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rw-r--r--	hoang	supergroup	22.95 KB	Dec 26 23:29	1	128 MB	prompts_hdfs.jsonl

Showing 1 to 1 of 1 entries Previous 1 Next

Hadoop, 2020.

Bước 4: Chuẩn bị model và code để chạy mapreduce

1. Tại model encoder từ link: https://huggingface.co/naver-iv/zim-anything-vitb/tree/main/zim_vit_b_2043



2. Xây dựng code Prompt_Encoder có tên là prompt_stage.py

```python
from __future__ import annotations
from dataclasses import dataclass
from typing import Optional, Tuple

import numpy as np
import torch

@dataclass
class PromptPack:
    # Tensor format đúng với predictor/model.decoder đang dùng
    points: Optional[Tuple[torch.Tensor, torch.Tensor]]  # (coords[B,N,2], labels[B,N])
    boxes: Optional[torch.Tensor]                        # [B,4] XYXY
    attn_mask: Optional[torch.Tensor]                    # [B,S,S] (nếu dùng)

class PromptStage:
    """
    'Prompt Encoder' theo nghĩa hệ thống (không phải class PromptEncoder của SAM):
    - transform point coords theo predictor.transform
    - convert sang torch tensor
    - tạo attention mask (nếu model yêu cầu)
    - trả về PromptPack cho decoder stage
    """

    def __init__(self, transform, model, device: torch.device):
        self.transform = transform  # từ predictor (ResizeLongestSide hoặc tương đương)
        self.model = model          # ZIM model wrapper (có point_attn_mask)
        self.device = device

    @torch.no_grad()
    def build(
```

```
        self,
        original_size: Tuple[int, int],
        point_coords: Optional[np.ndarray] = None,    # [N,2] pixel theo ảnh
gốc
        point_labels: Optional[np.ndarray] = None,    # [N]
        boxes: Optional[np.ndarray] = None,           # [4] hoặc [B,4] theo ảnh
gốc
    ) -> PromptPack:

        points_torch = None
        boxes_torch = None
        attn_mask = None

        # 1) Points
        if point_coords is not None:
            assert point_labels is not None, "point_labels must be supplied if
point_coords is supplied."

            # predictor.py đang làm apply_coords(point_coords, original_size)
            point_coords_t = self.transform.apply_coords(point_coords,
original_size)

            coords_torch = torch.as_tensor(point_coords_t,
dtype=torch.float32, device=self.device)
            labels_torch = torch.as_tensor(point_labels, dtype=torch.float32,
device=self.device)

            # predictor.py dùng shape [1,N,2] và [1,N]
            coords_torch = coords_torch[None, :, :]
            labels_torch = labels_torch[None, :]

            points_torch = (coords_torch, labels_torch)

            # 2) Attention mask (theo predictor.py:
self.model.point_attn_mask(point_coords))
            # Lưu ý: model.point_attn_mask expects point_coords [B,N,2]
            attn_mask = self.model.point_attn_mask(coords_torch)

        # 3) Boxes
        if boxes is not None:
            # hỗ trợ boxes dạng [4] hoặc [B,4]
            if boxes.ndim == 1:
                boxes = boxes[None, :]
            # transform boxes theo original_size (nếu predictor có
apply_boxes)
            if hasattr(self.transform, "apply_boxes"):
                boxes_t = self.transform.apply_boxes(boxes, original_size)
            else:
                # fallback: transform 2 điểm (x0,y0) và (x1,y1)
                # (chỉ dùng khi không có apply_boxes)
                boxes_t = boxes.copy()
```

```
                boxes_t[:, 0:2] = self.transform.apply_coords(boxes[:, 0:2],
original_size)
                boxes_t[:, 2:4] = self.transform.apply_coords(boxes[:, 2:4],
original_size)

            boxes_torch = torch.as_tensor(boxes_t, dtype=torch.float32,
device=self.device)

        return PromptPack(points=points_torch, boxes=boxes_torch,
attn_mask=attn_mask)
```

3. Xây dựng Transformer_Decoder có tên là decode_stage.py

```
import torch
import torch.nn.functional as F

class DecodeStage:
    def __init__(self, model):
        self.model = model

    @torch.no_grad()
    def run(
        self,
        image_embeddings,
        interm_feats,              # <-- BẮT BUỘC cho ZIM decoder
        points=None,
        boxes=None,
        attn_mask=None,
        multimask_output=False
    ):
        # ZIM decoder requires interm_feats
        masks, iou_predictions = self.model.decoder(
            image_embeddings=image_embeddings,
            interm_feats=interm_feats,
            points=points,
            boxes=boxes,
            attn_mask=attn_mask,
        )

        # low_res_masks: giữ tương tự predictor.py (upsample x2)
        low_res_masks = F.interpolate(
            masks,
            scale_factor=2,
            mode="bilinear",
            align_corners=False
        )

        # output dạng sigmoid như predictor
```

```
        return masks.sigmoid(), iou_predictions, low_res_masks.sigmoid()
```

4. Xây dựng Mapper để có thể chạy module với Mapreduce lấy tên là mapper_zim.py

```
import sys
import json
import os
import tempfile
import subprocess
import base64

import numpy as np
import cv2
import torch

from zim_anything import zim_model_registry, ZimPredictor
from zim_anything.modular.prompt_stage import PromptStage
from zim_anything.modular.decode_stage import DecodeStage


# ==================== HDFS HELPERS ====================

def hdfs_cmd() -> str:
    """
    Return path to hdfs executable.
    On Windows, prefer %HADOOP_HOME%\\bin\\hdfs.cmd
    """
    hadoop_home = os.environ.get("HADOOP_HOME") or
os.environ.get("HADOOP_PREFIX")
    if hadoop_home:
        cand = os.path.join(hadoop_home, "bin", "hdfs.cmd")
        if os.path.exists(cand):
            return cand
    # Fallback (if hdfs is in PATH)
    return "hdfs"


def hdfs_get(hdfs_path: str, local_path: str) -> None:
    """
    Download a file from HDFS to local_path.
    Supports hdfs:///path style by converting to /path.
    """
    if hdfs_path.startswith("hdfs:///"):
        hdfs_path = hdfs_path.replace("hdfs:///", "/")
    cmd = [hdfs_cmd(), "dfs", "-get", "-f", hdfs_path, local_path]
    subprocess.check_call(cmd, stdout=subprocess.DEVNULL,
stderr=subprocess.DEVNULL)
```

```python
# ==================== OUTPUT ENCODING ====================

def mask_to_png_b64(mask01: np.ndarray) -> str:
    """
    mask01: float/0-1 array -> PNG bytes -> base64 ascii
    """
    m = (mask01 > 0.5).astype(np.uint8) * 255
    ok, buf = cv2.imencode(".png", m)
    if not ok:
        raise RuntimeError("encode png failed")
    return base64.b64encode(buf.tobytes()).decode("ascii")


def emit(obj: dict) -> None:
    sys.stdout.write(json.dumps(obj, ensure_ascii=False) + "\n")
    sys.stdout.flush()


# ==================== MAIN ====================

def main():
    # ---- checkpoint dir (required) ----
    checkpoint_dir = os.environ.get("ZIM_CKPT_DIR")
    if not checkpoint_dir:
        raise RuntimeError(
            "Missing env var ZIM_CKPT_DIR. "
            "Set it to folder containing encoder.onnx and decoder.onnx, "
            "e.g. C:\\Users\\hoang\\Downloads\\BTL-
BigData\\ZIM\\results\\zim_vit_l_2092"
        )

    if not os.path.isdir(checkpoint_dir):
        raise RuntimeError(f"ZIM_CKPT_DIR is not a directory:
{checkpoint_dir}")

    enc_path = os.path.join(checkpoint_dir, "encoder.onnx")
    dec_path = os.path.join(checkpoint_dir, "decoder.onnx")
    if not os.path.exists(enc_path):
        raise RuntimeError(f"encoder.onnx not found in ZIM_CKPT_DIR:
{enc_path}")
    if not os.path.exists(dec_path):
        raise RuntimeError(f"decoder.onnx not found in ZIM_CKPT_DIR:
{dec_path}")

    # ---- device ----
    device = "cuda" if torch.cuda.is_available() else "cpu"

    # ---- build model ----
    keys = list(zim_model_registry.keys())
    if not keys:
        raise RuntimeError("zim_model_registry empty")
```

```
    model_key = keys[0]  # bạn có thể hardcode key cụ thể nếu muốn

    model = zim_model_registry[model_key](checkpoint=checkpoint_dir)
    model.to(device).eval()

    predictor = ZimPredictor(model)
    prompt_stage = PromptStage(
    predictor.transform,
    predictor.model,
    next(predictor.model.parameters()).device
)

    decode_stage = DecodeStage(predictor.model)

    # ---- streaming loop ----
    for line in sys.stdin:
        line = line.strip()
        if not line:
            continue

        try:
            rec = json.loads(line)

            rid = rec.get("id", "")
            hdfs_img = rec["image"]

            points = rec.get("points")          # [[x,y], ...] theo ảnh gốc
            point_labels = rec.get("point_labels")  # [1/0, ...]
            box = rec.get("box")                # [x1,y1,x2,y2] theo ảnh gốc

            # --- fetch image from HDFS to local temp ---
            with tempfile.TemporaryDirectory() as td:
                local_img = os.path.join(td, "img.jpg")
                hdfs_get(hdfs_img, local_img)

                bgr = cv2.imread(local_img)
                if bgr is None:
                    raise RuntimeError(f"cv2.imread failed: {local_img}")
                rgb = cv2.cvtColor(bgr, cv2.COLOR_BGR2RGB)

            # --- encoder stage ---
            # --- encoder stage ---
            predictor.set_image(rgb)  # tạo predictor.features + (thường)
interm feats
            image_embeddings = predictor.features
            interm_feats = get_interm_feats_from_predictor(predictor)

            if interm_feats is None:
                raise RuntimeError(
                    "Cannot find interm_feats on predictor after set_image().
"
```

```
                    "Add debug to print predictor attrs containing
'feat'/'inter'."
                )


            # --- prepare prompt inputs ---
            pc = pl = bx = None
            if points is not None and point_labels is not None:
                pc = np.array(points, dtype=np.float32)
                pl = np.array(point_labels, dtype=np.float32)
            if box is not None:
                bx = np.array(box, dtype=np.float32)

            # --- prompt stage ---
            pack = prompt_stage.build(
                original_size=predictor.original_size,
                point_coords=pc,
                point_labels=pl,
                boxes=bx,
            )

            # --- decode stage ---
            masks, iou, low_res = decode_stage.run(
                image_embeddings=image_embeddings,
                interm_feats=interm_feats,
                points=pack.points,
                boxes=pack.boxes,
                attn_mask=pack.attn_mask,
                multimask_output=False,
            )


            mask01 = masks[0, 0].detach().cpu().numpy()
            iou_score = float(iou[0, 0].detach().cpu().numpy())

            emit({
                "id": rid,
                "image": hdfs_img,
                "iou": iou_score,
                "mask_png_b64": mask_to_png_b64(mask01),
            })

        except Exception as e:
            emit({
                "id": rec.get("id", "") if isinstance(locals().get("rec"),
dict) else "",
                "error": str(e),
                "line": line[:300],
            })
```

```
def get_interm_feats_from_predictor(predictor):
    """
    Try to retrieve intermediate features produced by encoder.
    ZIM decoder requires these.
    """
    for name in ["interm_feats", "interm_features", "features_interm", "feat",
"feats", "interm"]:
        if hasattr(predictor, name):
            val = getattr(predictor, name)
            if val is not None:
                return val
    return None

if __name__ == "__main__":
    main()
```

5. Tạo file Reducer lấy tên là reducer_avg_iou.py

```
import sys, json

def main():
    sum_iou = 0.0
    count = 0
    min_iou = None
    max_iou = None

    # Streaming reducer input thường là: key \t value
    for line in sys.stdin:
        line = line.strip()
        if not line:
            continue

        try:
            # hỗ trợ cả 2 kiểu:
            # 1) "ALL\t{json...}"
            # 2) "{json...}"  (phòng khi mapper không emit key)
            if "\t" in line:
                _, v = line.split("\t", 1)
            else:
                v = line

            obj = json.loads(v)
            iou = float(obj.get("iou", 0.0))

            sum_iou += iou
            count += 1
            min_iou = iou if min_iou is None else min(min_iou, iou)
            max_iou = iou if max_iou is None else max(max_iou, iou)
```

```
        except Exception:
            # bỏ qua dòng lỗi
            continue

    avg = (sum_iou / count) if count else 0.0

    out = {
        "avg_iou": avg,
        "num_samples": count,
        "min_iou": min_iou,
        "max_iou": max_iou
    }

    # reducer output: 1 dòng JSON
    sys.stdout.write(json.dumps(out, ensure_ascii=False) + "\n")

if __name__ == "__main__":
    main()
```

Bước 5: Di chuyển đến thư mục chưa code để chuẩn bị chạy Map-Reduce

cd "C:\Users\hoang\Downloads\BTL-BigData\ZIM"

Bước 6: Thực hiện chạy Map-Reduce

Set env cho phiên PowerShell hiện tại

$env:ZIM_CKPT_DIR = "C:/Users/hoang/Downloads/BTL-BigData/ZIM/results/zim_vit_l_2092"

Xoá output cũ

Remove-Item -Recurse -Force .\mr_eval_out -ErrorAction SilentlyContinue

Đường dẫn hadoop + jar

$hadoop = Join-Path $env:HADOOP_HOME "bin\hadoop.cmd"

$jar = Join-Path $env:HADOOP_HOME "share\hadoop\tools\lib\hadoop-streaming-3.3.0.jar"

Chạy streaming local (MAP + REDUCE)

& $hadoop jar $jar `

-D mapreduce.framework.name=local `

-fs file:/// `

-input file:///C:/Users/hoang/Downloads/BTL-BigData/ZIM/prompts_local.jsonl `

-output file:///C:/Users/hoang/Downloads/BTL-BigData/ZIM/mr_eval_out `

-mapper "python mapper_zim.py" `

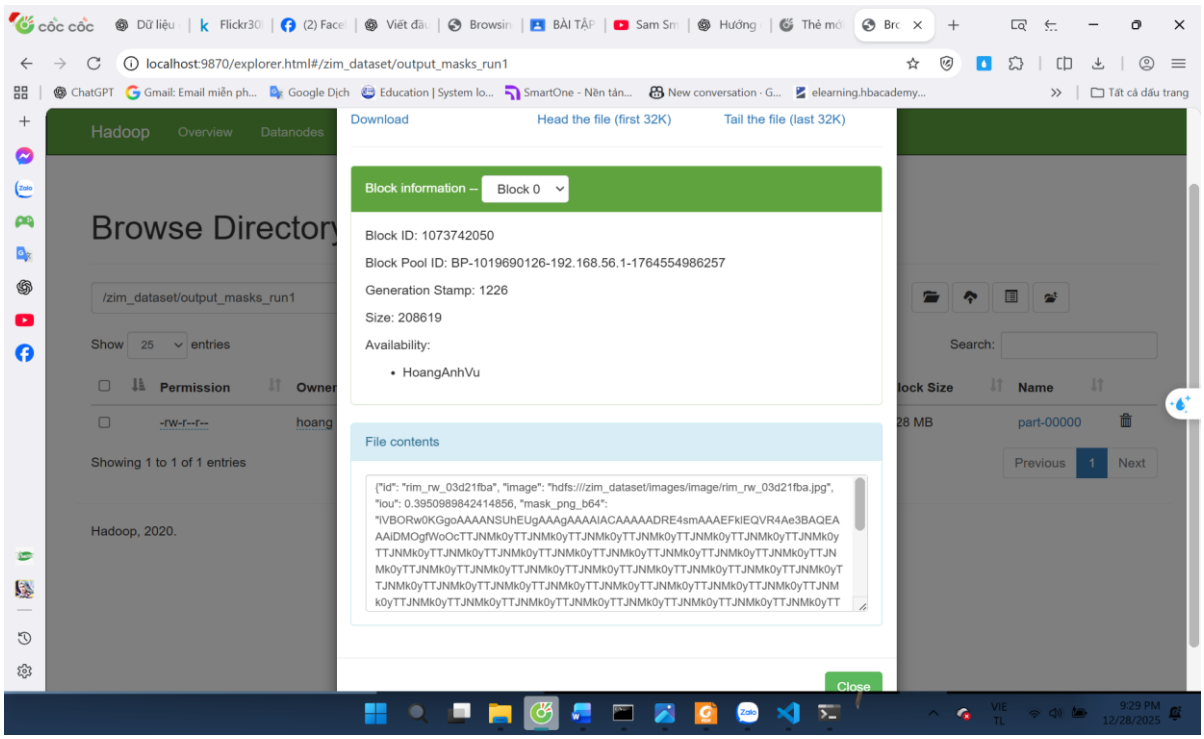
-reducer "python reducer_avg_iou.py" `

-file mapper_zim.py `

-file reducer_avg_iou.py `

-verbose

Bước 7: Kết quả thu được



Browse Directory

/zim_dataset/mr_eval_out Go!

Show 25 entries Search:

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rw-r--r--	hoang	supergroup	0 B	Dec 27 23:43	1	128 MB	_SUCCESS
-rw-r--r--	hoang	supergroup	116 B	Dec 27 23:43	1	128 MB	part-00000

Showing 1 to 2 of 2 entries Previous 1 Next