□ Data Preparation (Final Fix: Robust Validation + Local Time)

Features:

- **Thai Time:** Run ID uses GMT+7 (Thailand Time).
- **Strict Validation:** Skips corrupt/empty/unreadable images immediately.
- **Filter:** Excludes 'bark01' images.
- **HEIC Support:** Handles .heic files safely.

Path: /workspace/AiTaxonomy

In [ ]:
```
# Install HEIC library if missing (One-time setup)
import sys
import subprocess
try:
    import pillow_heif
except ImportError:
    subprocess.check_call([sys.executable, "-m", "pip", "install", "pillow-heif"])

import os
import glob
import random
import shutil
import numpy as np
import tensorflow as tf
from tqdm.notebook import tqdm
from datetime import datetime, timezone, timedelta
import cv2
from PIL import Image
import pillow_heif

# ================= CONFIGURATION =================
DATA_DIR = r"/workspace/Archive/All-Species"
OUTPUT_BASE_DIR = r"/workspace/AiTaxonomy/TFRecords_AllSpecies_B6"
LOG_DIR = r"/workspace/AiTaxonomy/TF-Training-Logs-B6"
```

```
IMG_SIZE = 528
VAL_SPLIT = 0.2
SEED = 123
IMAGES_PER_SHARD = 2000
# =================================================

os.makedirs(LOG_DIR, exist_ok=True)

def get_user_input(prompt):
    return input(prompt).strip()

def get_thai_timestamp():
    # Create GMT+7 Timezone
    tz_thai = timezone(timedelta(hours=7))
    return datetime.now(tz_thai).strftime("%Y%m%d-%H%M%S")

print(f"□ Configuration Loaded.")
```

In [ ]:

```
# ============================================================================
# □ STEP 1: Select ID (Thai Time)
# ============================================================================

existing_ids = sorted(os.listdir(OUTPUT_BASE_DIR)) if os.path.exists(OUTPUT_BASE_DIR) else []
print(f"□ Existing IDs: {existing_ids}")

user_id = get_user_input("Enter RUN ID to resume/overwrite (or press Enter for NEW): ")

if not user_id:
    RUN_TIMESTAMP = get_thai_timestamp()
    MODE = 'NEW'
    print(f"□ NEW ID (Thai Time): {RUN_TIMESTAMP}")
else:
    RUN_TIMESTAMP = user_id
    target_dir = os.path.join(OUTPUT_BASE_DIR, RUN_TIMESTAMP)
    if os.path.exists(target_dir):
        print("1) Resume  2) Overwrite  3) Cancel")
        choice = get_user_input("Select: ")
        if choice == '1': MODE = 'RESUME'
        elif choice == '2': MODE = 'OVERWRITE'
        else: MODE = 'CANCEL'
    else:
        MODE = 'NEW'
```

In [ ]:

```
# ==============================================================================
# ▯ STEP 2: Strict Processing Functions
# ==============================================================================

def _bytes_feature(value):
    if isinstance(value, type(tf.constant(0))):
        value = value.numpy()
    return tf.train.Feature(bytes_list=tf.train.BytesList(value=[value]))

def _int64_feature(value):
    return tf.train.Feature(int64_list=tf.train.Int64List(value=[value]))

def serialize_example(image_string, label):
    feature = {
        'image': _bytes_feature(image_string),
        'label': _int64_feature(label),
    }
    return tf.train.Example(features=tf.train.Features(feature=feature)).SerializeToString()

def process_image_safely(img_path, target_size):
    """Reads image with strict checks. Returns bytes if valid, None if corrupt."""
    try:
        # 1. Check File Existence & Size
        if not os.path.exists(img_path) or os.path.getsize(img_path) == 0:
            print(f"× Skipping Zero-byte/Missing file: {img_path}")
            return None

        ext = os.path.splitext(img_path)[1].lower()
        img = None

        # 2. Decode Image
        if ext in ['.heic', '.heif']:
            heif_file = pillow_heif.read_heif(img_path)
            image = Image.frombytes(heif_file.mode, heif_file.size, heif_file.data, "raw")
            img = np.array(image)
            img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
        else:
            # Use byte stream to avoid path encoding issues
            with open(img_path, "rb") as stream:
                bytes_data = bytearray(stream.read())
                numpyarray = np.asarray(bytes_data, dtype=np.uint8)
                img = cv2.imdecode(numpyarray, cv2.IMREAD_COLOR)

        # 3. Validate Content
        if img is None:
            print(f"× Skipping Corrupt Image (Decode Failed): {img_path}")
            return None

        if img.size == 0 or img.shape[0] == 0 or img.shape[1] == 0:
            print(f"× Skipping Empty Dimensions: {img_path}")
            return None

        # 4. Resize & Encode
        img = cv2.resize(img, (target_size, target_size))
        is_success, img_encoded = cv2.imencode('.jpg', img, [int(cv2.IMWRITE_JPEG_QUALITY), 95])

        if not is_success:
            print(f"× Skipping Encode Error: {img_path}")
            return None

        return img_encoded.tobytes()

    except Exception as e:
        print(f"× Error processing {img_path}: {e}")
        return None

def write_tfrecords_robust(data, output_dir, prefix, class_map, resume=False):
    if not os.path.exists(output_dir): os.makedirs(output_dir)

    start_index = 0
    shard_idx = 0

    # Clean Resume: Delete last shard to prevent corruption
    if resume:
        files = sorted(glob.glob(os.path.join(output_dir, f"{prefix}_*.tfrecord")))
        if files:
            last_file = files[-1]
            try:
                os.remove(last_file)
                print(f"⚠ Removed last shard {last_file} to ensure data integrity.")
            except: pass
            shard_idx = len(files) - 1
            start_index = shard_idx * IMAGES_PER_SHARD
            print(f"□ Resuming from index {start_index} (Shard {shard_idx})")

    if start_index >= len(data):
        print(f"□ {prefix} already complete.")
        return

    writer = None
    data_to_process = data[start_index:]

    print(f"Processing {len(data_to_process)} candidates for {output_dir}...")

    valid_count = 0

    for i, img_path in tqdm(enumerate(data_to_process), total=len(data_to_process)):
        # Rotate shard
        if valid_count % IMAGES_PER_SHARD == 0 and valid_count > 0:
            pass

        # Simple Sharding Logic
        if writer is None or (i % IMAGES_PER_SHARD == 0):
            if writer: writer.close()
            shard_path = os.path.join(output_dir, f"{prefix}_{shard_idx:04d}.tfrecord")
            writer = tf.io.TFRecordWriter(shard_path)
            shard_idx += 1

        class_name = os.path.basename(os.path.dirname(img_path))
        label = class_map.get(class_name)

        if label is not None:
            img_bytes = process_image_safely(img_path, IMG_SIZE)
            if img_bytes:
                writer.write(serialize_example(img_bytes, label))
                valid_count += 1
    if writer: writer.close()
    print(f"□ {prefix} Done. Wrote {valid_count} valid images (Skipped {len(data_to_process) - valid_count} bad fil
```

In [ ]:

```
# ==============================================================================
# □ STEP 3: EXECUTE
# ==============================================================================

if MODE != 'CANCEL':
    SAVE_DIR = os.path.join(OUTPUT_BASE_DIR, RUN_TIMESTAMP)

    if MODE == 'OVERWRITE':
        print(f"□ Deleting old data in {SAVE_DIR}...")
        shutil.rmtree(SAVE_DIR)

    # 1. File Scanning
    print("□ Scanning files...")
    classes = sorted([d for d in os.listdir(DATA_DIR) if os.path.isdir(os.path.join(DATA_DIR, d))])
    class_map = {name: i for i, name in enumerate(classes)}

    all_files = []
    valid_ext = {'.jpg', '.jpeg', '.png', '.bmp', '.webp', '.heic', '.heif'}

    skipped_bark = 0

    for cls in tqdm(classes):
        cls_path = os.path.join(DATA_DIR, cls)
        if os.path.exists(cls_path):
            for f in os.listdir(cls_path):
                if os.path.splitext(f)[1].lower() in valid_ext:
                    if 'bark01' in f.lower():
                        skipped_bark += 1
                    else:
                        all_files.append(os.path.join(cls_path, f))

    print(f"⚠ Skipped {skipped_bark} 'bark01' images.")

    # 2. Shuffle & Split
    random.seed(SEED)
    random.shuffle(all_files)
    val_count = int(len(all_files) * VAL_SPLIT)
    train_files = all_files[val_count:]
    val_files = all_files[:val_count]

    print(f"□ Total Candidates: {len(all_files)} (Train: {len(train_files)}, Val: {len(val_files)})")

    # 3. Process with Validation
    is_resume = (MODE == 'RESUME')
    write_tfrecords_robust(train_files, os.path.join(SAVE_DIR, 'train'), 'train_data', class_map, resume=is_resume)
    write_tfrecords_robust(val_files, os.path.join(SAVE_DIR, 'val'), 'val_data', class_map, resume=is_resume)

    print(f"\n□ READY FOR TRAINING. ID: {RUN_TIMESTAMP}")
```